# Privacy Compliance Verification in Cryptographic Protocols

Suriadi Suriadi[1], Chun Ouyang[1,2], and Ernest Foo[1]

[1] Science and Engineering Faculty, Queensland University of Technology, Australia
{s.suriadi, c.ouyang, e.foo}@qut.edu.au
[2] NICTA, Queensland Research Laboratory, Brisbane, Australia

**Abstract.** To provide privacy protection, cryptographic primitives are frequently applied to communication protocols in an open environment (e.g. the Internet). We call these protocols privacy enhancing protocols (PEPs) which constitute a class of cryptographic protocols. Proof of the security properties, in terms of the privacy compliance, of PEPs is desirable before they can be deployed. However, the traditional provable security approach, though well-established for proving the security of cryptographic primitives, is not applicable to PEPs. We apply the formal language of Coloured Petri Nets (CPNs) to construct an executable specification of a representative PEP, namely the Private Information Escrow Bound to Multiple Conditions Protocol (PIEMCP). Formal semantics of the CPN specification allow us to reason about various privacy properties of PIEMCP using state space analysis techniques. This investigation provides insights into the modelling and analysis of PEPs in general, and demonstrates the benefit of applying a CPN-based formal approach to the privacy compliance verification of PEPs.

## 1 Introduction

To achieve privacy-enhancing features, cryptographic primitives employed in a privacy enhancing protocol (PEP) normally have rich features (e.g. verifiable encryption) which extend the common encryption and signature capabilities often used in other types of *cryptographic protocols* (e.g. authentication protocols). For example, emulating the off-line anonymity afforded by cash transactions, a PEP can ensure that when a user purchases goods on-line, the on-line seller does not learn the identity of the user, but at the same time can be assured that the user's identity has been previously verified by a known trusted entity such that the identity can be revealed when needed. Recently, the Trusted Platform Module (TPM) technology - which provides secure hardware storage of cryptographic keys and implementation of cryptographic primitives - has also been used in PEPs [24].

An important issue in the design of applied cryptographic protocols, such as PEPs, is to ensure that they work correctly and do not contain errors that may weaken the security protection provided by the cryptographic primitives employed. While the *provable security* approach [15] is a widely-accepted method

used to prove the security properties of cryptographic primitives, it is not suitable to verify privacy compliance properties of PEPs. The main reason is that provable security emphasizes on proving the properties of a cryptographic algorithm (as evidenced by the use of ideal cryptographic models, such as the random oracle model - see [5, 18]), while the privacy compliance properties of a PEP are *behavioural* and can be more naturally reasoned as properties of communication protocols. For example, one of the privacy properties verified in this paper is the *enforceable conditions* property (detailed in Sect. 4.3). This property is concerned with whether the messages exchanged between protocol entities are such that enough safeguards are included to ensure that a user's PII is *indeed* only revealed when certain conditions are satisfied, even in the presence of malicious behaviours from the entities involved. Consequently, attacks in PEPs normally arise from the existence of multi-party entities who attempt to exploit weaknesses in the *design* of a protocol, not directly at the algorithms of the cryptographic primitives employed. Furthermore, due to the lack of computer-aided tool support, the provable security approach is prone to errors [17].

Formal methods and languages allow the construction of unambiguous and precise models that can be analysed to identify errors and to verify correctness before implementation. Some of them, such as Coloured Petri Nets (CPNs) [14], provide a graphical modelling capability, and have tool support. The application of formal methods has been demonstrated to lead to reliable and trustworthy security protocols [2, 8]. However, to the best of our knowledge, verification of PEPs using formal methods is yet to mature.

In this paper, we propose a CPN-based approach to construct a formal specification of a representative PEP, namely the Private Information Escrow Bound to Multiple Conditions Protocol (PIEMCP) [21], and to verify its privacy compliance properties.[3] CPNs are a widely-used formal language for system specification, design, simulation and verification. CPNs provide a graphical modelling language capable of expressing concurrency and system concepts at different levels of abstraction. With the support of CPN Tools, basic constructs of Petri nets are enriched with the functional programming language Standard ML (SML) [13] such that various high-level data type definition and functions can be defined and used in the model.

PIEMCP involves non-trivial multi-party communication (6 or more entities in general) and employs complex cryptographic primitives and TPM functionalities. The hierarchical structuring mechanism of CPNs supports a modular approach in capturing the behaviour of PIEMCP at different levels of abstraction. Using SML, the essential structures and behaviours of a wide variety of privacy-enhancing cryptographic primitives can be captured through a "black-box-style" abstraction such that only the essential features remain. By parameterising the protocol model with different types of attacks, a large number of attack scenarios are captured for analysis. The CPN model of PIEMCP is executable and can be analysed to verify the privacy properties of the protocol using the state spaces generated from the parameterised CPN model.

[3] This paper is an extension from our earlier work [22].

The contributions of this paper are two-fold. Firstly, it demonstrates the use of a CPN-based approach to model and verify the privacy properties of a PEP. To our knowledge, our work so far has been the only attempt at the formal verification of PEPs using CPNs. Secondly, the paper proposes several modelling and analysis approaches that have been (or can be) applied to other PEPs [20,24]. These can be used as preliminary guidelines for a general CPN-based approach for modelling and verification of PEPs.

This paper is structured as follows. Sect. 2 briefly explains PIEMCP. Sect. 3 proposes the modelling approach and describes selected parts of the CPN model of PIEMCP. Based on this CPN model, Sect. 4 details the verification of PIEMCP focusing on a set of privacy compliance properties. Related work is discussed in Sect. 5 with conclusions provided in Sect. 6. We assume that the reader has basic knowledge of CPNs. While we endeavour to explain the basic idea of PIEMCP, given the space constraints, prior knowledge in the area of information security and privacy is useful.

## 2 Overview of PIEMCP

PIEMCP [21] is used in a federated single-sign on (FSSO) environment whereby a user only has to authenticate once to an identity provider (IdP) to access services from multiple service providers (SPs). The entities involved are users, IdPs, SPs, and an anonymity revocation manager (ARM) or referees. An IdP assures SPs that although users are anonymous, when certain conditions are fulfilled, the users' identity can be revealed. A user's identity refers to a set of personally identifiable information (PII). Although the services that SPs provide can be delivered without the need of PII, they require the PII to be revealed by an ARM *or* referees when certain *conditions* are satisfied. An example of *conditions* would be "the user $X$'s PII should only be revealed to $SP1$ if the user has posted some inflammatory/illegal messages/pictures on the forum".

PIEMCP consists of four stages, namely PII escrow (PE), key escrow (KE), multiple conditions (MC) binding, and revocation. An execution of the protocol involves two distinct sessions: the *escrow session* which consists of a sequential execution of the PE, KE and MC stages, and the *revocation session* which consists of an execution of the revocation stage. A user can run $n$ escrow sessions, during which his/her PII is hidden (anonymous). At least one escrow session has to be completed before a revocation session can start. During the revocation session, the user's PII linked to a specific SP in a specific escrow session is revealed. For $n$ escrow sessions, each with $m$-number of SPs, up to $n \times m$ revocation sessions can be performed.

PIEMCP has two variants: one uses a trusted ARM for anonymity revocation while the other uses a group of referees instead of ARM. While these two variants do overlap to a certain degree, in this paper, we only consider the second variant of PIEMCP because it involves concurrent behaviours which highlight the relevance of CPN as the modelling language. Figure 1 depicts the *main* message exchanges between the different entities of this protocol. For simplicity,

a double-arrowed line is an abstraction of an exchange of one or more messages which collectively achieve a single cryptographic operation (normally a proof-of-knowledge operation).

The *PE stage* begins when a user requests a service from a service provider SP1. This triggers the agreement between user and SP1 of *conditions* (denoted $Cond1$) whose fulfillment allows the PII to be revealed to SP1, and a set of UCHVE parameters (explained in the ensuing paragraphs). SP1 then sends a message `NT-PE-1` containing $Cond1$ and UCHVE parameters to the IdP to escrow the user's PII. The IdP contacts the user to obtain his encrypted PII (NT-PE-2). The user then encrypts his PII using a Verifiable Encryption (VE) scheme under a freshly generated public ($\texttt{pubk}_{VE}$) and private ($\texttt{privk}_{VE}$) key pair. The output of this VE operation is a ciphertext denoted as $\texttt{VE(PII)}_{pub_{VE}}$. The user sends to the IdP, `NT-PE-3` which comprises of $\texttt{VE(PII)}_{pub_{VE}}$ and $\texttt{pubk}_{VE}$. The user keeps $\texttt{privk}_{VE}$ which is needed to decrypt $\texttt{VE(PII)}_{pub_{VE}}$. Next, the user and the IdP engage in a cryptographic "proof-of-knowledge" (PK) protocol (`NT-PE-4`). This is to prove to the IdP that the VE ciphertext given correctly hides some *certified* PII without letting the IdP learn the value of the PII itself. We denote this operation as `PKVE`. The output of PKVE is an acceptance or rejection of $\texttt{VE(PII)}_{pub_{VE}}$. A successful PKVE operation will lead to the IdP generating and sending a `pseudonym` to the user (`NT-PE-5`).

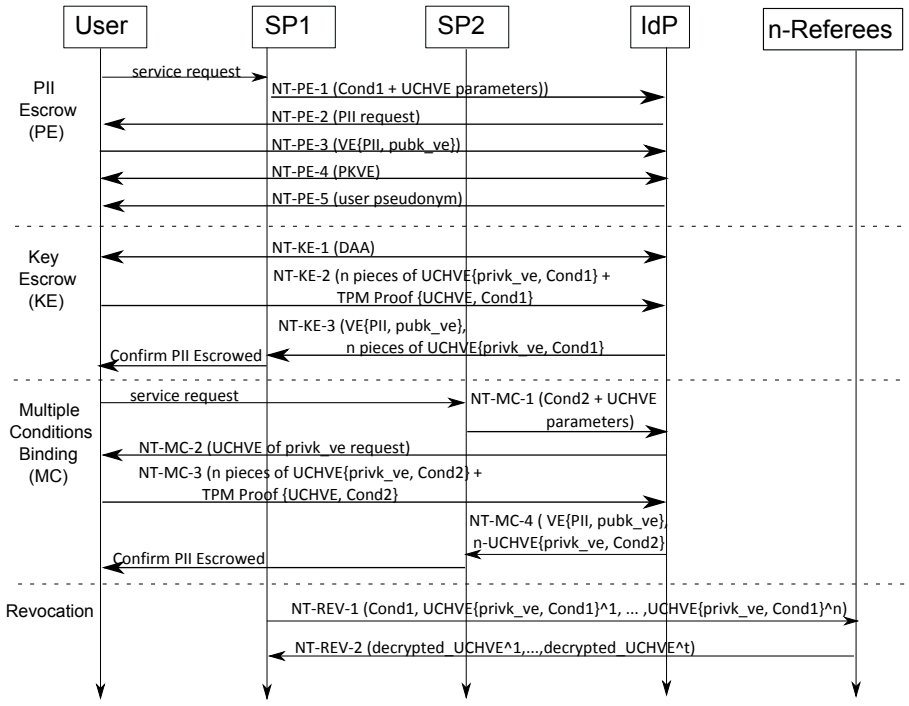


Fig. 1: Message exchanges within the four stages of PIEMCP.

The *KE stage* is started after the user receives and stores the `pseudonym`. The IdP and the user now engage in another PK protocol - the Direct Anonymous Attestation (DAA) (`NT-KE-1`). This is to convince the IdP that the user is using a valid TPM device while concealing the identity of the TPM device. A successful DAA prompts the user's TPM to generate (1) a universal custodian-hiding verifiable group encryption (UCHVE) of $\texttt{priv}_{VE}$ under $Cond1$ (denoted as $\texttt{UCHVE(privk}_{VE}\texttt{)}^{1...n}_{Cond1}$ and (2) a TPM proof of a correct UCHVE execution. In the rest of this paper, the generation of such UCHVE ciphertext with the TPM proof are represented by a TPM module, called `TPM Module 2`. The $\texttt{UCHVE(privk}_{VE}\texttt{)}^{1...n}_{Cond1}$ actually is a group of $n$ distinct ciphertext pieces which can later be given to a group of $n$ referees among whom there are $t$ ($t \leq$n) *designated* referees. Only the designated referees can decrypt their respective ciphertext pieces. At least $k$ ($k \leq t$) decrypted pieces are required to recover the VE private key (i.e. $k$ is the *threshold* value). Both $\texttt{UCHVE(privk}_{VE}\texttt{)}^{1...n}_{Cond1}$ and the corresponding TPM proof are sent to the IdP in `NT-KE-2`. The IdP then verifies the proof and if correct, prepares a response `NT-KE-3` to SP2 which includes the $\texttt{VE(PII)}_{pub_{VE}}$ and $\texttt{UCHVE(privk}_{VE}\texttt{)}^{1...n}_{Cond1}$. Having obtained $\texttt{VE(PII)}_{pub_{VE}}$ and $\texttt{UCHVE(privk}_{VE}\texttt{)}^{1...n}_{Cond1}$, SP1 now can, with the help of referees, recover the user's PII when $Cond1$ is fulfilled, but *cannot* do so until that time. SP1 then confirms to the user that his/her PII has been escrowed successfully.

In the *MC stage*, the user goes to another service provider SP2 to request service. This time SP2 requests the IdP to escrow the $\texttt{privk}_{VE}$ in `NT-MC-1` under *conditions* $Cond2$ (i.e. $Cond1 \neq Cond2$) and UCHVE parameters that have been agreed between user and SP2. The IdP requests the user's TPM to produce $\texttt{UCHVE(privk}_{VE}\texttt{)}^{1...n}_{Cond2}$ (that is, a *new* UCHVE encryption of $\texttt{privk}_{VE}$ under $Cond2$) and the associated TPM proof (`NT-MC-2`). The user then performs the requested operation and sends $\texttt{UCHVE(privk}_{VE}\texttt{)}^{1...n}_{Cond2}$ with the corresponding TPM proof in `NT-MC-3` (in other words, the `TPM Module 2` is executed again). The IdP verifies the TPM proof of $\texttt{UCHVE(privk}_{VE}\texttt{)}^{1...n}_{Cond2}$, and if correct, prepares a response `NT-MC-4` to SP2 which includes $\texttt{VE(PII)}_{pub_{VE}}$ and $\texttt{UCHVE(privk}_{VE}\texttt{)}^{1...n}_{Cond2}$. Similar to SP1, SP2 now has the necessary ciphertexts which, with referees' help, can reveal the user's PII when $Cond2$ are satisfied, but *cannot* do so yet at this point. SP2 then confirms to the user that his/her PII has been escrowed successfully.

For any subsequent service providers that the user contacts within an escrow session, the user and the service provider only need to execute the MC stage activities. Therefore, the MC stage activities are specific to the *second and subsequent* service providers visited by the user, while the PE and KE stage activities are specific to only the *first* SP visited by the user.

The *revocation stage* is executed when the agreed conditions are satisfied and when a user has completed at least one escrow session. Assuming that $Cond1$ is satisfied, SP1 sends a revocation request $\texttt{NT-REV-1}_{1...n}$ to each of the $n$ referees. For each referee $r_i$, the message $\texttt{NT-REV-1}_i$ consists of $\texttt{UCHVE(privk}_{VE}\texttt{)}^{i}_{Cond1}$ and $Cond1$. Each referee then checks if $Cond1$ is fulfilled, and if so, the referee tries to decrypt the given ciphertext piece. Only the designated referees can de-

crypt the ciphertext pieces. If decryption is successful, each designated referee sends the decrypted data $\texttt{NT-REV-2}_{1...t}$ to SP1. When $k$ ($k \leq t$) or more decrypted data are received, SP1 can recover $\texttt{priv}_{VE}$, and subsequently decrypt $\texttt{VE(PII)}_{pub_{VE}}$ to recover the user's PII.

Above we described the normal execution of PIEMCP (i.e. without attacks). However, each of the parties involved may behave maliciously resulting in various attack scenarios. The design goal of PIEMCP is to achieve the required security behaviour with and without considering the attacks. In the next section, a CPN model of PIEMCP is presented which can be configured to capture both normal scenario and attack scenarios.

## 3 CPN Model of PIEMCP

### 3.1 modelling Approach

We introduce two modelling approaches that are specific to PEPs: the *Cryptographic Primitive Abstraction* and the *Model Parameterization with Attacks*. We have also captured the *TPM Provable Execution* modelling approach in our model but it is not described in this paper (the details are available at [19, Section F.2]).

*Cryptographic Primitive Abstraction.* To capture complex cryptographic behaviours, we firstly model the representation of a ciphertext as a CPN colour set, and then capture its operations using SML functions. This approach is flexible and inclusive as virtually any type of cryptographic primitives can be captured. The CPN `record` type can encode the necessary information to represent a primitive properly, and SML can be used to *simulate* the operations. The cryptographic operations captured by SML functions are "symbolic" rather than an actual operation. For example, an encryption function defined in our approach *does not* perform the actual encryption, rather, we impose certain restriction on what the recipient of this ciphertext can do with this message (such as not being able to extract the message without having a correct decryption key).

Our approach of expressing cryptographic operations as functions promotes reuse which leads to a cleaner and more concise model. However, a disadvantage of this approach is that the modeler has to consciously follow the restriction imposed on cryptographic messages produced by these functions as CPN Tools does not automatically enforce these restrictions. In Sect. 3.2, we demonstrate this approach by modelling a VE ciphertext and a zero-knowledge operation (PKVE). The complexity of UCHVE ciphertext prevents us from describing it due to space constraint; however, it is available in the referenced thesis [19, pp. 196].

We also propose a technique to capture the commonly-used message signing and verification operations. We define a CPN colour set for the message to be signed, followed by a definition of its signature. A signed message is a pair consisting of the message and its signature. The verification of a signed message upon the receipt of the message is *enforced* within a transition guard.

If the signature verification fails, the message integrity and/or authenticity are compromised. As a result, the guard returns a false value, thus preventing any further processing on the message – a so-called fail-stop mechanism.

*Model Parameterization with Attacks.* We propose the parameterisation approach to modelling attacks such that one or more attacks can be switched on or off depending on the environmental assumptions. At this point, we scope our work to only consider malicious insiders - which we consider to be a greater concern in PEPs. The Dolev-Yao intruder model [12] (which represents an external intruder), while relevant, is not considered in this paper. There are many attacks that a malicious insider could launch. Creating a new model to capture each type of attack (existing or new) scales poorly as the number of attacks grows. Parameterisation allows the re-use of the existing model while allowing it to behave differently according to the attacks being set - virtually allowing thousands of possible attack scenarios to be captured. We have modelled 14 types of attacks using 14 parameters, each with a boolean value of "true" (on) or "false" (off), which theoretically can capture $2^{14}$ possible attack scenarios. The attack parameters are then referred to in the arc-inscriptions, transition guards, or transition code-regions. Note that although it is not necessary to consider all attack scenarios (see Sect. 4), the ability of our model to capture a comprehensive attack scenario may be exploited in the future to allow other types of analysis.

The advantage of this approach is that we do not have to change the model (e.g. adding/deleting transitions) to obtain different behaviours. The disadvantage however is that it may reduce the readability of the model due to the addition of parameter inscriptions (such as `if/else` statements) and may make model debugging more difficult as the number of attacks increases. This approach risks the introduction of complexity during model validation in comparison to having two separate models (one without attacks and one with attacks). However, this risk is somewhat compensated with an easier model maintenance practice: changes to the basic behaviour of the model only need to be applied once to the model and its effect will apply to all other parameterized behaviours. This is not the case when we have two or more separate models.

### 3.2 Model Description

The PIEMCP model is a hierarchical CPN consisting of 4 levels: 1 main (top-level) page, 5 second-level pages, 13 third-level pages, and 1 fourth-level page. As detailed in Sect. 2, a sequential execution of the PE, the KE, and the MC stage forms one *escrow session*. For simplicity, our model covers a minimum full protocol execution: the PIEMCP CPN model allows sequential execution of a certain number of escrow session (determined by the model parameter `session`) followed by one revocation session. Note that it is possible for both the escrow and revocation session to run in parallel, however, modelling such concurrency does not capture any additional behaviour of the protocol as these two sessions

are assumed to be distinct, i.e. they do not interfere with each other.[4] Therefore, our model currently does not capture this parallelism.

The revocation page can be executed after the completion of at least *one* escrow session. Selected parts of the PIEMCP CPN model are described to demonstrate the modelling approaches detailed in Sect. 3.1. Relevant CPN colour sets definitions and functions are provided in Table 1.

*Main page.* Figure 2 shows the top-level page which captures the protocol entities (represented as *substitution transitions*) and the communication channels between any two entities (as *places* with thick lines). Since these communication channels represent application-layer communication, we assume the existence of no errors commonly associated with lower-layer communication channels (such

[4] While it may be interesting to model and analyze the security properties of our protocol in the presence of parallel *escrow* and *revocation* sessions, we reserve this for future work.

```
1  CRYPTOGRAPHIC COLOUR SETS DEFINITION
2  ====================================
3  colset K_PUB_VE = INT;
4  colset K_PRIV_VE = INT;
5  colset K_SIGN_GEN = INT;
6  colset PII = STRING;
7  colset LABEL = STRING;
8  colset PROVABILITY = BOOL;
9  colset COMMITMENT_PII = record message:PII * random:RANDOM;
10 colset CIPHER_VE_PII = record message:PII * key:K_PUB_VE * label:LABEL*provable:PROVABILITY;
11
12 PIEMCP MESSAGES DEFINITION
13 ==========================
14 colset SP_REQ = record genCond:STRING * conditions1:STRING * <other fields omitted>
15 colset SP_REQ_SIG = record message:SP_REQ * key:K_SIGN_GEN;
16 colset SIGNED_SP_REQ = record message:SP_REQ * signat:SP_REQ_SIG;
17 colset SIGNATURE_GEN = record message:MSG * key:K_SIGN_GEN * provable: PROVABILITY;
18 colset SIGNED_MSG = record message:MSG * signat:SIGNATURE_GEN;
19 colset DEC_REQ = record conditions:LABEL * uchvePiece:CIPHER_UCHVE_KVE_PIECE;
20 colset DEC_REQ_SIGNATURE = record message:DEC_REQ * key:K_SIGN_GEN * provable:BOOL;
21 colset SIGNED_DEC_REQ = record message:DEC_REQ * signat:DEC_REQ_SIGNATURE;
22 colset DECRYPT_OUTPUT = product BOOL * MSG;
23
24 COMMUNICATION CHANNEL DEFINITION
25 ================================
26 colset IDP_SP1 = union msgEscrow:SIGNED_SP_REQ + signedSPResponse1:SIGNED_SP_RESPONSE;
27
28 FUNCTIONS and PARAMETERS
29 ========================
30 fun veKeysRel(privKey:K_PRIV_VE, pubKey:K_PUB_VE)= if privKey=pubKey then true else false;
31 fun veEnc(msg:PII, pubKey:K_PUB_VE,cond1:LABEL)=
32     {message=msg,key=pubKey,label=cond1,provable=true};
33 fun decVE(key:K_PRIV_VE, cipherVE:CIPHER_VE_PII, cond1:LABEL)=
34     if veKeysRel(key, #key(cipherVE)) andalso cond1 = (#label(cipherVE)) then
35     1`(true, #message(cipherVE)) else 1`(false, "");
36 val condActually = true;
37 val session=2;
38 val threshold=2;
39 val honestRef=1;
40 val toRevoke=1;
```

Table 1: Colour set definition

as data loss). While it may be possible to fold the three `SP1_REFEREE` channels into one, we decided to split them into three to improve readability (i.e. to explicitly separate distinct logical communication channels between entities).

As explained in Sect. 2, the PE and KE stage activities are specific to the *first* service provider (e.g. SP1) visited by the user, while the MC stage activities are specific to the *second and subsequent* service providers (e.g. SP2). Therefore, we decided to separate the modelling of SP activities into two substitution transitions (SP1 and SP2) due to their *non-overlapping* activities.

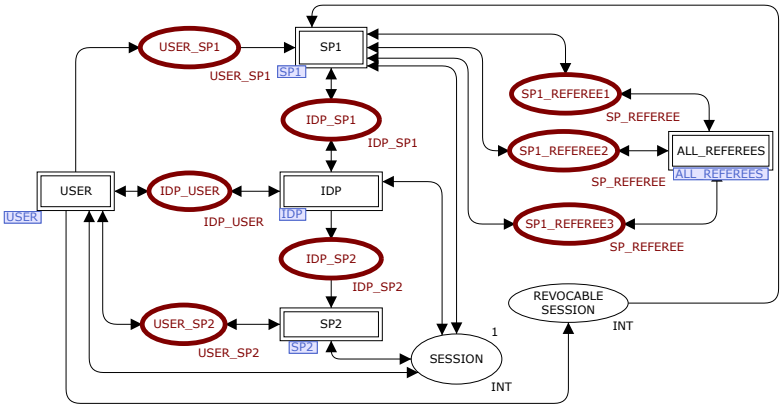


Fig. 2: The PIEMCP CPN – Top-level page

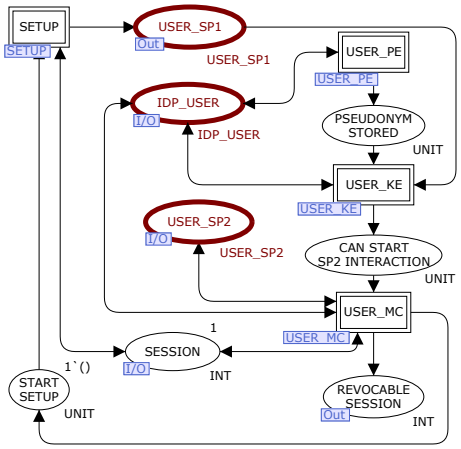


Fig. 3: The PIEMCP CPN – USER second-level page

*Second-level Pages.* The multi-stage operation of PIEMCP is detailed on the second-level pages for each of the entities. For example, the second-level page for

the user is shown in Figure 3 whereby the sequential execution of the PE, KE (with SP1), and MC (with SP2) stages is modelled. The completion of the MC stage signals the completion of one escrow session and may trigger the execution of another escrow session by marking the place `START_SETUP`. The determination of whether or not to execute another escrow depends on the model parameter `session` and is explained in detail in the Appendix A.2. Furthermore, the completion of an escrow session also means that, theoretically, the user PII for that particular escrow session is now revocable. However, as explained in the beginning of Sect. 3.2, our model will only allow the execution of the revocation stage after the completion of a certain number of escrow sessions as determined by the parameter `session`. Similar second-level pages for IdP, SP1, and SP2 have also been modelled (detailed in Appendix A.1).

*Third-level Pages.* The details of the IdP's, SPs', and user's activities during the PE, KE, MC, and revocation stages are provided on the third-level pages. Four examples of such pages are provided in Figures 4 to 7. Note that the main transitions in Figures 4 to 6 have been annotated with a number (located at the bottom-right corner of each of the transitions) indicating the normal order (i.e. all attack parameters switched off) in which they occur.

Figure 4 depicts the model of the IdP's activities during the PE stage. This page demonstrates the message signing and verification approach. The input arc to the transition `IDP_VERIFIES_SP1_REQ_AND_STARTS_PII_ESCROW` (Figure 4, top centre) contains a variable `escrowReqSig` of colour set `SIGNED_SP_REQ` within the `union` colour set `IDP_SP1` (see Table 1 lines 14, 15 and 26). The `escrowReqSig` variable represents an SP1-signed message whose content is the *conditions* string. This message is equivalent to message `NT-PE-1` in Figure 1. As the IdP receives this message, it verifies the signature validity which is captured in the transition guard of the same transition. If the guard expression (`verifyEscrowReqSig(escrowReqSig)`) returns true, the signature is valid and the transition is enabled, allowing the IdP to contact the user to proceed with the PE stage.

Figure 5 depicts the details of the user's activites. This page models the generation of necessary cryptographic data by the user. Here, we demonstrate how complex cryptographic primitive behaviours can be modelled. The VE ciphertext is defined as the colour set `CIPHER_VE_PII` (see Table 1 line 10) which is a record consisting of four fields: the message itself, the public encryption key, the label under which the message is encrypted, and the provability property. A provable ciphertext means that the recipient of the ciphertext can validate that the received ciphertext correctly encrypts some claimed value (in this case the user's PII) without the recipient learning the value of either the PII itself or the decryption key. We consider the `message` field inside a colour set that represents a ciphertext to be *unreadable*. The model in Figure 5 captures the generation of a VE ciphertext of PII, the result of which will trigger the placement of a token in the `PII_VE_CIPHER` place (Figure 5, top-right).

The VE operations, including the encryption and decryption operations, are captured as functions (see Table 1 lines 30-35). As stated in Sect. 3.1, our encryption operation does not perform the actual message encryption and decryption

operation. Rather, these operations are abstracted into two functions – `veEnc` and `decVE` – and an auxilliary function `veKeysRel`. The function `veEnc` transforms the main inputs for a VE encryption algorithm and outputs a token typed by the colour set `CIPHER_VE_PII`. The decryption operation (1) takes as input a representation of a VE private key and the ciphertext to be decrypted, (2)

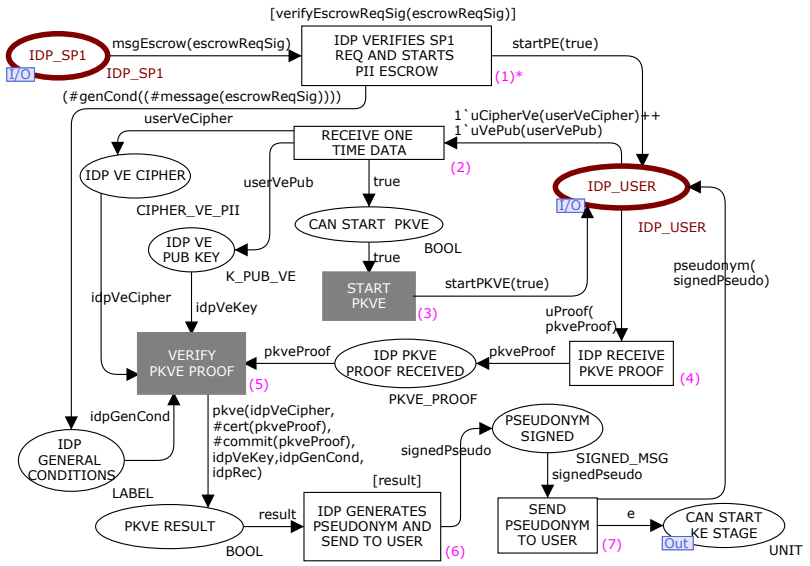


Fig. 4: The PIEMCP CPN – IDP PE page.

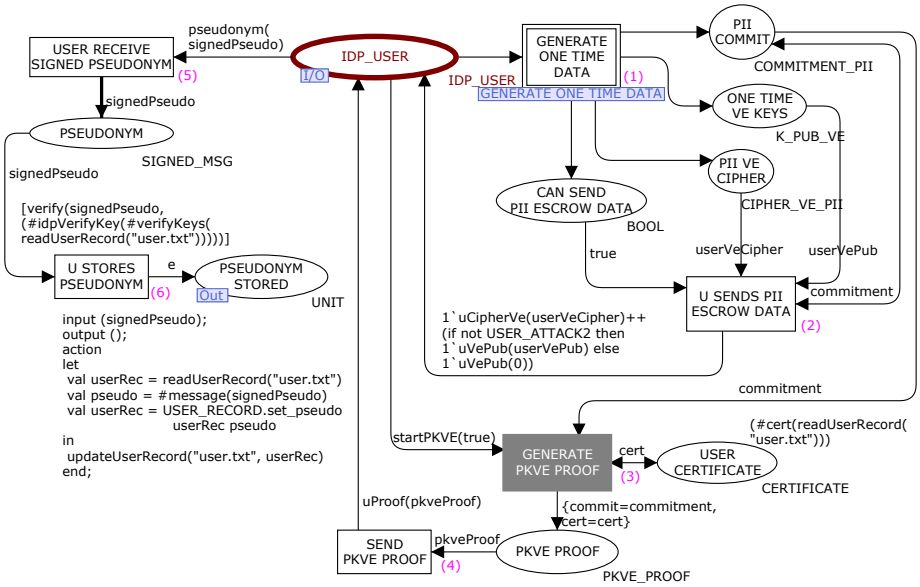


Fig. 5: The PIEMCP CPN – USER PE page

checks if the private key and the public key used to produce the ciphertext satisfies some *relation* (as captured by `veKeysRel`) and if the condition string given is indeed the same as the one used for producing the ciphertext, then it outputs the result (3) in the form of the `DECRYPT_OUTPUT` colour set (Table 1 line 22): the first element indicates the success/failure of the decryption, and the second element contains the decrypted message (in case of success).

Next, the user sends the `NT-PE-3` message (containing the VE ciphertext of PII, and the public VE key) to the IdP - represented by the transition `U_SENDS_PII_ESCROW_DATA` (Figure 5, middle-right). When the IdP receives this message, the `PKVE` operation is triggered (`NT-PE-4`). Here, we demonstrate how a complex zero-knowledge proof protocol like `PKVE` is modelled in CPN. We break this operation into three transitions across the `USER_PE` and the `IDP_PE` pages (indicated with grey-filled transitions): the `START_PKVE` transition triggered by IdP to signal the user the start of such a protocol (Figure 4, centre), the `GENERATE_PKVE_PROOF` transition executed on the user side to generate the required `PKVE` proof data (Figure 5, middle-bottom), and the `VERIFY_PKVE_PROOF` transition executed by the IdP to verify the given `PKVE` proof data (Figure 4, middle-left). The result of `PKVE` is represented by the place `PKVE_RESULT`. The essential processing required by the IdP to verify the correctness of the proof is captured by the function `pkve`[5], which is invoked in the arc inscription from the transition `VERIFY_PKVE_PROOF` to the place `PKVE_RESULT`. Upon a successful `PKVE`, the IdP generates a pseudonym and sends it to the user to be used for that particular session.

Figure 5 also shows the attack parameterisation approach mentioned in Sect. 3.1. The `USER_ATTACK2` parameter (see the output arc inscription from the transition `U_SENDS_PII_ESCROW_DATA` to the place `IDP_USER` around the centre of Figure 3) depicts the behaviour of a malicious user who falsifies/gives an incorrect VE public key to the IdP in the `NT-PE-2` message. Thus, when `USER_ATTACK2` is set to "true", the user will send an incorrect VE public key value (represented as "0"), otherwise, a correct value is sent.

Figure 6 shows the model for the revocation stage. The first transition (top) `SP1_RETRIEVES_FULFILLED_CONDITIONS` is only enabled if the total number of executed escrow sessions (represented by the variable `counter`) is greater than the value of the parameter `session`. The completion of an escrow session increases the value of `counter` by one, and as a result, the completion of `session`-number of escrow sessions will result in the value of `counter` to be `session+1`. Hence, the guard to the above transition essentially ensures that there must be at least `x`-number of escrow sessions completed before the revocation stage can start (whereby `x` is determined by the `session` parameter).

Next, SP1 firstly retrieves the condition string of a completed escrow session which is deemed to have been fulfilled (the model parameter `toRevoke` - Table 1 line 40 - determines the corresponding completed escrow session). This data is retrieved from the stored session data executed through the code segment associated with the transition `SP1_RETRIEVES_FULFILLED_CONDITIONS` (which is not

[5] Definition of this function is available in the referenced thesis [19, pp. 305].

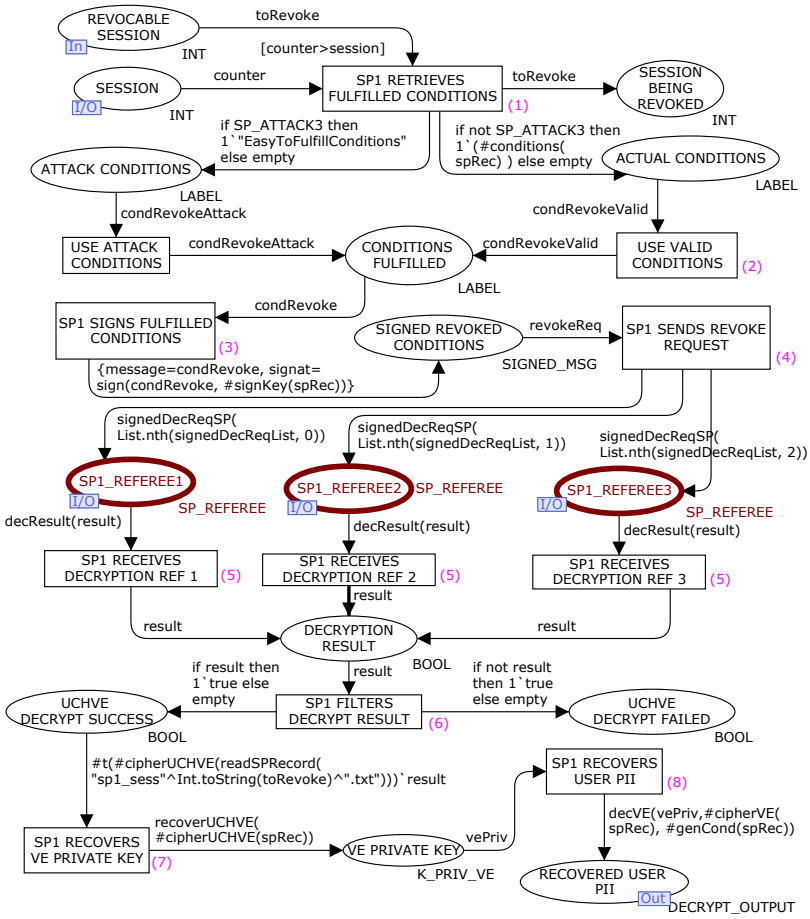


Fig. 6: The PIEMCP CPN – SP1_REV page (revocation stage initiated by SP1)

shown in Figure 6). Note that the session data were previously stored by SP1 at the completion of the KE stage (details available in Appendix A.4). Figure 6 also demonstrates the parameterisation of another attack parameter `SP_ATTACK3` whereby a service provider may attempt to revoke a user's PII by including the trivial "easy to fulfill" condition string (which is different from what was agreed with the user previously during the escrow stage). This is captured in the two output arcs from the transition `SP1_RETRIEVES_FULFILLED_CONDITIONS`. The setting of `SP_ATTACK3` will mark the place `ATTACK_CONDITIONS` (Figure 6, top left) with the "easy to fulfill" condition string and no token will be sent to the place `ACTUAL_CONDITIONS` (Figure 6, top right). In the absence of this attack, the place `ACTUAL_CONDITIONS` will be marked with the actual condition string as read from the session data file and the place `ATTACK_CONDITIONS` will not have any token.

SP1 then sends a PII revocation request to all referees, modelled by the transition `SP1_SENDS_REVOKE_REQUEST` (Figure 6, middle right), by sending the condition string and the UCHVE pieces which are retrieved by reading the session

data stored previously (achieved through code-region not shown in Figure 6). The details of each referee's model are described later in this section; at this stage, when the SP1 receives the decrypted UCHVE pieces, it will then attempt to recover the VE private key. This is captured by the input arc to the transition `SP1_RECOVERS_VE_PRIVATE_KEY` (Figure 6, bottom left). This arc inscription requires $t$ (representing the threshold value) successfully decrypted pieces of the UCHVE ciphertext by referees before the message (i.e. the VE private key) can be decrypted. This page also demonstrates how CPN can be used to capture the concurrent processing required (amongst the referees) during the UCHVE decryption process. The combination of the modelling approach used on this page and the referee pages (desribed in the ensuing text) therefore demonstrates how we can captures a *threshold decryption* process using CPN.

The details of the referees' model are described below. Figure 7 (left-hand side) shows the detailed referees' activities during the revocation stage. Since the operations of each referee are the same, we decided to create one `REFEREE` page which can be instantiated for individual referees. An example of a `REFEREE` page instance is shown in Figure 7 (right-hand side). To capture the different runtime behaviour of individual referees, we parameterise each `REFEREE` page instance (on the `ALL_REFEREE` page) with two main parameters: the referee number ($ID$) and the condition fulfillment decision (the `REFEREE_NUMBER_`$i$ and `COND_FULFILLMENT_`$i$ places respectively, where $i$={1,2,3}). The later parameter is used to capture the (non-)malicious behaviour of a referee and is determined through the setting of its initial value. For example, the initial marking of `COND_FULFILLMENT_1` (Figure 7, top left) states that when all attack parameters which affects the referees'

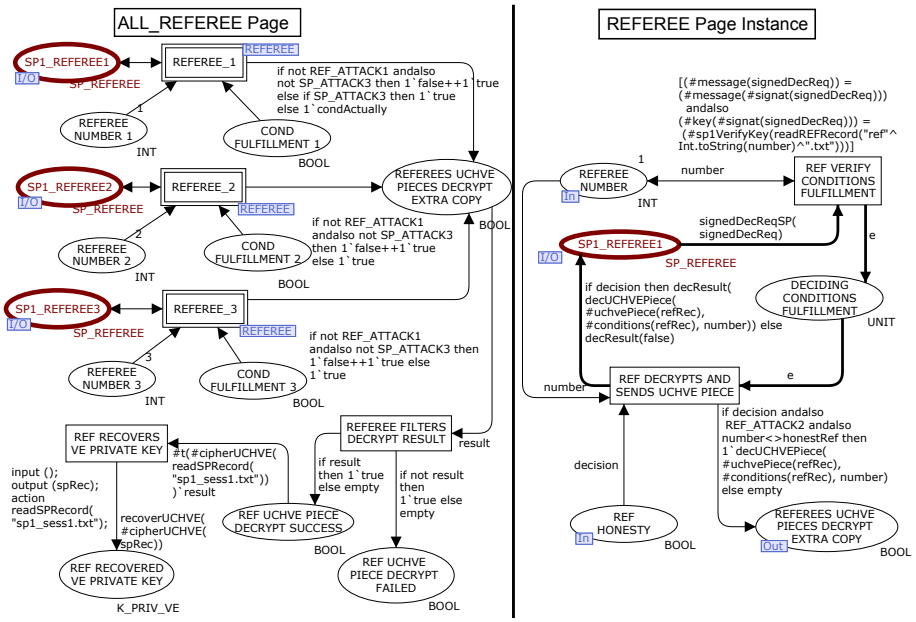


Fig. 7: ALL_REFEREES page (left) and a REFEREE page instance (right)

decision of conditions fulfillment (i.e. REF_ATTACK1 and SP_ATTACK3) are false, the model considers both situations whereby the 1st Referee (REFEREE_1) agrees and disagrees to the fulfillment of the conditions; hence the place is initialized with both a true token and a false token.

When any one of the attacks listed above is set to true, we now need to be able to differentiate between "honest" referee and "malicious" referee. The REF_ATTACK1 parameter captures the behaviour of malicious referees which agree on some conditions fulfillment even when it is not the case. An honest referee will state the *actual* fulfillment of the conditions, hence, for REFEREE_1 (which represents an honest referee - consistent with our protocol assumption that there exists at least one honest designated referee), the value of the place COND_FULFILLMENT_1 (Figure 7, top left) is set according to the model parameter condActually. For malicious referees (REFEREE_2 and REFEREE_3), the initial value of their corresponding places, COND_FULFILLMENT_2 and COND_FULFILLMENT_3 (Figure 7, left middle), will be set to true (when REF_ATTACK1 is set to true) to capture the malicious behaviour decsribed before. The attack parameter SP_ATTACK3 is defined to capture the behaviour of a service provider attempting to launch a revocation session using a set of made-up conditions which will most likely cause the referees to agree to their fulfillment. When this parameter is switched on, the initial marking for the above-mentioned places of all referees is a true token.

Figure 7 (right) also shows the parameterization of malicious referees who attempt to pool all decrypted UCHVE pieces amongst themselves with the hope of being able to recover the VE private key (captured by the parameter REF_ATTACK2). By studying the inscription of the arc from the transition REF_DECRYPTS_AND_SENDS_UCHVE_PIECE to place REFEREES_UCHVE_DECRYPT_EXTRA_COPY (Figure 7, bottom right) and by observing the parameter honestRef set to 1 (Table 1 line 39), this malicious behaviour only applies to REFEREE_2 and REFEREE_3.

## 4 Verification of PIEMCP

### 4.1 Analysis Approach

We conduct the verification of PIEMCP using state space analysis. The verification can be complex due to the numerous avenues by which attackers could attempt to break the privacy protection provided by PIEMCP. We propose to scope the verification within a set of plausible known attack scenarios.

The verification of PIEMCP takes into account both the absence and presence of attack behaviours, and is carried out in two stages: the baseline behaviour analysis (Sect. 4.2) and privacy compliance verification (Sect. 4.3). Firstly, the baseline behaviour analysis is performed through standard state space analysis, including the inspection of proper session termination, deadlock/livelock freedom, and absence of unexpected dead transitions. As a result, the analysis informs us about the baseline correctness of PIEMCP. Next, we specify a set of common privacy compliance properties of PIEMCP using ASK-CTL [10], a dialect of Computational Tree Logic (CTL), supported by CPN Tools. These

property statements are then interpreted into queries for model-checking the state spaces generated from the PIEMCP CPN model to prove if the privacy compliance holds for the protocol.

The PIEMCP CPN model has an initial state where the protocol can begin the setup process and the session number is initialised with the first session's identifier and the three referee identifiers are specified. This is captured by the initial marking $M_0$ where the places `START_SETUP` and `SESSION` on the `USER` page (Figure 3, bottom) and the three `REFEREE_NUMBER_`$i$ ($i = 1, 2, 3$) places on the `ALL_REFEREES` page (Figure 7, left-hand side) are marked accordingly. Furthermore, our model has a `session` parameter to execute two sequential *escrow sessions* before a revocation is started (see Table 1 line 37).

In the absence of attack behaviour (that is, all attack parameters are set to `false`), the state space generated from the above configuration has 606 nodes and 1374 arcs, and contains no cycles (given the fact that the SCC graph has the same number of nodes and arcs). The PIEMCP CPN model is then configured to include a number of known attacks, and is executed under each of the configurations. A set of state spaces is generated capturing the behaviour of PIEMCP with the corresponding attack scenarios.

We introduce some notations to be used. $CPN_P^{M_0}$ denotes the PIEMCP CPN model with an initial marking $M_0$. $P^{\texttt{PageName}}_{\texttt{PlaceName}}$ and $T^{\texttt{PageName}}_{\texttt{TransitionName}}$ refer to a specific place and transition in the CPN model, respectively. The marking of a place is then written as $M(P^{\texttt{PageName}}_{\texttt{PlaceName}})$.

### 4.2 Baseline Behaviour Analysis

The standard state space report generated from $CPN_P^{M_0}$ *without* any attack behaviour (when all the attack parameters are set to `false`) shows that there are 8 dead markings. A close inspection of these markings indicates that they reflect all 8 different protocol termination points based on the dynamic conditions fulfillment decision (boolean decision) by the 3 referees modelled on the `ALL_REFEREES` page (Figure 7). Also, there are three dead transitions: $T^{\texttt{SPI\_REV}}_{\texttt{USE\_ATTACK\_COND}}$ (Figure 6), $T^{\texttt{ALL\_REFEREES}}_{\texttt{REFEREE\_FILTERS\_DECRYPT\_RESULT}}$, and $T^{\texttt{ALL\_REFEREES}}_{\texttt{REF\_RECOVERS\_VE\_PRIVATE\_KEY}}$ (Figure 7-left). These are expected dead transitions because they reflect attack behaviours.

Moreover, the report shows that both the upper and the lower integer bounds of the place $P^{\texttt{USER}}_{\texttt{SESSION}}$ is 1 (i.e. a place invariant). This is expected since the place is marked with the identifier of an ongoing escrow session throughout the protocol execution (where sessions are executed one by one without interruption). Also, the place $P^{\texttt{ALL\_REFEREES}}_{\texttt{REF\_RECOVERED\_VE\_PRIVATE\_KEY}}$ is always *empty*, which is expected as it reflects the modelling strategy for capturing problems (which then marks this place) with the basic design of the PIEMCP itself. In conclusion, the state space report confirms the expected baseline behaviour of PIEMCP *without* attacks.

For the PIEMCP *with* attacks (when one or more attack parameters are set to `true`), the expected baseline behaviour is to stop the protocol execution as soon as an attack is detected - a *fail-stop* behaviour. We would like to validate that the PIEMCP CPN model exhibits such behaviour when taking into account all possible attack scenarios.

Table 2 details all the 14 attack parameters considered, and the particular stage of PIEMCP where each attack may take place. As mentioned in Sect. 3, the PIEMCP CPN model captures sequential executions of the four stages in the order that PE is followed by KE, then MC and optionally Revocation stage at last. Following this order, we first allow only the attacks to occur in the PE stage. There are $2^3-1$ attack scenarios resulting from combinations of 3 attack parameters (`USER_ATTACK1`, `USER_ATTACK2` and `SP_ATTACK1`). These are captured by 7 configurations of $CPN_P^{M_0}$ which then lead to the generation of 7 state spaces. Analysis of these state spaces shows that the protocol detects the above attacks and terminates within the PE stage. Similarly, we allow only the attacks to occur in the subsequent KE, MC, and Revocation stages, respectively, and the analysis results show that for each of the stages the protocol detects the relevant attacks and terminates within that stage.

From the above analysis, it follows that due to the sequential execution of the four stages of PIEMCP and the fact that the fail-stop mechanism *does* work within each of these stages, once an attack occurs in an earlier stage (e.g. PE) the protocol terminates within that stage, regardless of whether or not the attacks are allowed to happen in a subsequent stage (e.g. KE, MC, or Revocation). Therefore, the total 44 attack scenarios that pass the above fail-stop behaviour validation cover the behaviour of all possible $2^{14}$ attack scenarios in PIEMCP based on the list of 14 attack parameters specified in Table 2.

### 4.3 Privacy Compliance Verification

We define four privacy compliance properties for PIEMCP. These are formalised as ASK-CTL statements over $CPN_P^{M_0}$. CPN Tools support ASK-CTL [10] as an implementation of a subset of CTL (mainly the "until" operator) over the state spaces of CPN models. ASK-CTL implements two basic path quantification operators to capture this logic: EXIST_UNTIL($A_1$, $A_2$) and FORALL_UNTIL($A_1$, $A_2$). The EXIST_UNTIL operator means that there must be at least *one* path, from a given state, whereby predicate $A_1$ holds for every state in the path until the state where predicate $A_2$ holds. The FORALL_UNTIL operator is similar, except that it requires *all* paths to fulfill $A_1$ until $A_2$ is true. From these, two derived path quantification operators are POS($A$)=EXIST_UNTIL(`true`, $A$) and EV($A$)=FORALL_UNTIL(`true`, $A$), which check the reachability of a state in which predicate $A$ holds. More specifically, POS($A$) checks if there is at least *one* path that leads to a state where $A$ holds (i.e. it is **pos**sible to reach such a state), while EV($A$) checks if *all* paths lead to a state where $A$ holds (i.e. it must **ev**entually reach such a state).[6]

Below, we use the above ASK-CTL temporal operators, a dialect of those in CTL, to specify four privacy compliance properties in the context of PIEMCP. We introduce some notations to be used in the property definitions. Firstly, we divide the 14 attack parameters into two groups: $A_{ses}$ for the set of attack

[6] ASK-CTL provides many other operators, which we do not use in the compliance property specification.

| Entity | Parameter | Escrow Session | | | Revc. | Description |
|---|---|---|---|---|---|---|
| | | PE | KE | MC | | |
| User | USER_ATTACK1 | T | | | | Incorrect PII and *Cond1* used to generate $\texttt{VE(PII)}_{pub_{VE}}$ |
| | USER_ATTACK2 | T | | | | Incorrect $\texttt{pubk}_{VE}$ sent to IdP |
| | USER_ATTACK3 | | T | | | Incorrect UCHVE parameters used |
| | USER_ATTACK4 | | T | | | Non-agreed "hard to fulfill" conditions used in TPM Module2 |
| Service Provider | SP_ATTACK1 | T | | | | Non-agreed "easy to fulfill" conditions forwarded to IdP |
| | SP_ATTACK11 | | T | | | during PE and KE respectively |
| | SP_ATTACK12 | | T | | | Non-agreed UCHVE parameters forwarded to IdP |
| | SP_ATTACK2 | | | T | | Non-agreed "easy to fulfill" conditions forwarded to IdP at MC |
| | SP_ATTACK22 | | | T | | Non-agreed UCHVE parameters forwarded to IdP at MC |
| | SP_ATTACK3 | | | | T | See explanation for Figure 6, pp. 12 |
| | SP_ATTACK6 | | | T | | SP2 uses invalid signature key |
| | SP_ATTACK7 | | | T | | SP1 and SP2 use the same condition within an escrow session |
| Referee | REF_ATTACK1 | | | | T | See explanation for Figure 6 and |
| | REF_ATTACK2 | | | | T | Figure 7, pp. 12 |
| Number of attack scenarios to consider | | $2^3$-1 | $2^4$-1 | $2^4$-1 | $2^3$-1 | |

Table 2: The set of attack parameters, their effects on the PIEMCP stages, and the number of attack scenarios to consider.

parameters targeting an escrow session, and $A_{rev}$ for the set of attack parameters targeting a revocation stage. More specifically,

- $A_{ses} = \{$USER_ATTACK1, USER_ATTACK2, USER_ATTACK3, USER_ATTACK4, SP_ATTACK1, SP_ATTACK11, SP_ATTACK12, SP_ATTACK2, SP_ATTACK22, SP_ATTACK6, SP_ATTACK7$\}$
- $A_{rev} = \{$SP_ATTACK3, REF_ATTACK1, REF_ATTACK2$\}$

Next, we define two predicates with respect to an escrow session or a revocation:

- $S$ is the set of escrow sessions, $\forall s \in S$, $\mathsf{Session}^s M = (M(P^{\texttt{Main}}_{\texttt{SESSION}}) = 1`s)$
- $R$ is the set of revocable sessions, $\forall r \in R$, $\mathsf{Revoking}^r M = (M(P^{\texttt{SP1\_REV}}_{\texttt{BEING\_REVOKED}}) = 1`r)$

We refer to various places and transitions in the formalization of properties. Given the space constraints, only the formalization of the *enforceable conditions* can be followed using the CPN pages that have been described in Sect. 3.2. Other properties refer to certain places/transitions located within those CPN pages which are described in the Appendix.

**Multiple Conditions.** In PIEMCP, when no attack occurs during an escrow session, the *multiple conditions* property requires that the protocol always reaches the end of the session, and also each SP should receive an escrowed PII that is cryptographically bound to conditions which are different from one SP to another. Any attacks which may comprimise this property must be detected and

caused a premature ending of the protocol. We have configured the CPN model with one attack parameter that may compromise this property, `SP_ATTACK7`, which depicts the scenario of SPs colluding to use the same condition string with the same user in a session. The goal of this attack is to make sure that all SPs involved in an escrow session share the same *condition* string such that when it is satisfied, all SPs within that escrow session are authorized to learn the user's PII. Therefore, in such a scenario, we expect the user to detect it and prematurely end its interaction with the malicious SP.

We formalize the above informal property definition as follows: In $CPN_P^{M_0}$, in the absence of attack behaviour, when the protocol runs to the end of an escrow session, the place $P^{\texttt{USER\_MC}}_{\texttt{CAM\_START\_NEXT\_SESSION}}$ is marked signaling the end of a MC stage (i.e. the end of a session - see Figure 16 in Appendix A.5), and the place $P^{\texttt{Main}}_{\texttt{SESSION}}$ (Figure 2) is marked by the session identifier of that escrow session. The two places, $P^{\texttt{SP1\_KE}}_{\texttt{UCHVE\_COND}}$ (Appendix - Figure 15) and $P^{\texttt{SP2}}_{\texttt{UCHVE\_COND}}$ (Appendix - Figure18), which are used to store the above conditions regarding an escrowed PII for $SP_1$ and $SP_2$ respectively, should be marked by different conditions at the end of an escrow session. Informally, this means that the value of $Cond1$ and $Cond2$ (referred to in Section 2) must not be the same ($Cond1 \neq Cond2$).

When `SP_ATTACK7` is switched on, the desired behaviour of our protocol (reflecting the non-violation of this property) is captured by those execution paths which lead to a marking where $P^{\texttt{USER\_MC}}_{\texttt{CAN\_REQUEST\_SP2\_SERVICE}}$ (Appendix - Figure 16) is marked with a `false` token.

*Property 1 (**Multiple Conditions**).* With the following predicates:

- $\mathsf{SessionEnd}\ M = (M(P^{\texttt{USER\_MC}}_{\texttt{CAM\_START\_NEXT\_SESSION}}) = 1`e)$
- $\mathsf{DiffCondSP}\ M = (M(P^{\texttt{SP1\_KE}}_{\texttt{UCHVE\_COND}}) \neq \emptyset \wedge M(P^{\texttt{SP2}}_{\texttt{UCHVE\_COND}}) \neq \emptyset \wedge M(P^{\texttt{SP1\_KE}}_{\texttt{UCHVE\_COND}}) \neq M(P^{\texttt{SP2}}_{\texttt{UCHVE\_COND}}))$
- $\mathsf{ReqSP2Fail}\ M = (M(P^{\texttt{USER\_MC}}_{\texttt{CAN\_REQUEST\_SP2\_SERVICE}}) = 1`false)$

PIEMCP has *multiple conditions* property *iff* $CPN_P^{M_0}$ has the following behaviour:

- if all the attack parameters $A_{ses}$ are `false`, then
  $\forall s \in S, \forall M \in M_0 >: \textsc{Ev}(\mathsf{Session}^s\ M \wedge \mathsf{SessionEnd}\ M \wedge \mathsf{DiffCondSP}\ M)$
- otherwise, if `SP_ATTACK7` (and others in $A_{ses}$ are `false`), then
  $\exists s \in S, \exists M \in M_0 >: \textsc{Ev}(\mathsf{Session}^s\ M \wedge \mathsf{ReqSP2Fail}\ M)$ □

**Zero-knowledge.** In PIEMCP, when there are no attacks during an escrow session, and before the revocation of a user's PII for that escrow session, the zero-knowledge property requires that the IdP must validate that the ciphertexts (and the corresponding parameters) it possesses are correct while at the same time does not learn the value of the user's PII. When there are attacks which may compromise this property, we require our protocol to be able to detect it. A malicious entity (such as user) may falsify the ciphertexts or their related parameters with the hope that the IdP does not detect it and still accepts the

ciphertexts and the related parameters. If this situation occurs, then this property is violated due to flaws in the design of our protocol. We have modelled six attacks that may compromise this property with the parameters: `USER_ATTACK1`, `USER_ATTACK2`, `USER_ATTACK3`, `USER_ATTACK4`, `SP_ATTACK12`, and `SP_ATTACK22`. All of these attacks involve either the user or SPs sending to the IdP some incorrect/falsified ciphertexts and/or related parameters. For example, `USER_ATTACK1` involves the user sending to the IdP a ciphertext which encrypts some "garbage" data. The details of how we model these attacks are available in Appendix B (among which `USER_ATTACK2` was described in Sect. 3.2).

We formalize this property as follows: In $CPN_P^{M_0}$, three places, $P^{\texttt{IDP\_PE}}_{\texttt{PKVE\_RESULT}}$ (Figure 4, bottom-left corner), $P^{\texttt{IDP\_KE}}_{\texttt{TPM\_PROOF\_VERIFICATION\_RESULT}}$ (Figure 14, bottom-left), and $P^{\texttt{IDP\_MC}}_{\texttt{TPM\_PROOF\_VERIFICATION\_RESULT}}$ (Figure 17, bottom-middle), capture the correctness of the encryption. When there are no attacks, all three places must be marked by a `true` token; when any of the above-mentioned attacks is switched on, at least one of the three places must be marked by a `false` token.

*Property 2 (**Zero-knowledge**).* With the following predicates:

- UsrVE-T $M = (M(P^{\texttt{IDP\_PE}}_{\texttt{PKVE\_RESULT}}) = 1`true)$
- UsrVE-F $M = (M(P^{\texttt{IDP\_PE}}_{\texttt{PKVE\_RESULT}}) = 1`false)$
- UchveKE-T $M = (M(P^{\texttt{IDP\_KE}}_{\texttt{TPM\_PROOF\_VERIFICATION\_RESULT}}) = 1`true)$
- UchveKE-F $M = (M(P^{\texttt{IDP\_KE}}_{\texttt{TPM\_PROOF\_VERIFICATION\_RESULT}}) = 1`false)$
- UchveMC-T $M = (M(P^{\texttt{IDP\_MC}}_{\texttt{TPM\_PROOF\_VERIFICATION\_RESULT}}) = 1`true)$
- UchveMC-F $M = (M(P^{\texttt{IDP\_MC}}_{\texttt{TPM\_PROOF\_VERIFICATION\_RESULT}}) = 1`false)$

PIEMCP has *zero-knowledge* property *iff* $CPN_P^{M_0}$ has the following behaviour:

- if all the attack parameters in $A_{ses}$ are `false`, then $\forall s \in S$:
  $\text{Ev}(\mathsf{Session}^s \wedge \mathsf{UsrVE\text{-}T} \wedge \mathsf{UsrTPM\text{-}T} \wedge \mathsf{UchveKE\text{-}T} \wedge \mathsf{UchveMC\text{-}T}) \wedge$
  $\neg\text{Pos}(\mathsf{Session}^s \wedge (\mathsf{UsrVE\text{-}F} \vee \mathsf{UsrTPM\text{-}F} \vee \mathsf{UchveKE\text{-}F} \vee \mathsf{UchveMC\text{-}F}))$
- if `USER_ATTACK1` $\vee$ `USER_ATTACK2` (and others in $A_{ses}$ are `false`),
  then $\exists s \in S$: $\text{Ev}(\mathsf{Session}^s \wedge \mathsf{UsrVE\text{-}F}) \wedge \neg\text{Pos}(\mathsf{Session}^s \wedge \mathsf{UsrVE\text{-}T})$
- if `USER_ATTACK3` $\vee$ `USER_ATTACK4` $\vee$ `SP_ATTACK12` (and others in $A_{ses}$ are `false`),
  then $\exists s \in S$: $\text{Ev}(\mathsf{Session}^s \wedge \mathsf{UchveKE\text{-}F}) \wedge \neg\text{Pos}(\mathsf{Session}^s \wedge \mathsf{UchveKE\text{-}T})$
- if `SP_ATTACK22` (and others in $A_{ses}$ are `false`),
  then $\exists s \in S$: $\text{Ev}(\mathsf{Session}^s \wedge \mathsf{UchveMC\text{-}F}) \wedge \neg\text{Pos}(\mathsf{Session}^s \wedge \mathsf{UchveMC\text{-}T})$ □

**Enforceable Conditions.** The *enforceable conditions* property requires that a user's PII should never be revealed unless all designated referees agree that the cryptographically bound conditions are satisfied and that the referees must not be able to learn the value of the PII themselves (they can only decrypt UCHVE ciphertext pieces which does not allow them to learn the PII - at least $k$ decrypted UCHVE pieces are needed). This requirement applies regardless of whether there are any attack behaviours or not. Possible attacks that can be launched to compromise this property include those parameterised by `REF_ATTACK1` and `REF_ATTACK2` (both attacks have been described in Sect. 3.2).

We formalize this property as follows. Note that the fulfilment status of certain revocation conditions for a session is captured by parameter `condActually` (Table 1 line 36). In $CPN_P^{M_0}$, if `condActually` does not hold, then: (1) the number of decrypted UCHVE pieces ($|M(P^{\texttt{SP1\_REV}}_{\texttt{UCHVE\_DECRYPT\_SUCCESS}})|$ in Figure 6, bottom left) by the designated referees must be fewer than the minimum number of referees ($k$) needed for a successful PII revocation, and (2) the user PII must not be revealed by checking the marking which indicates the revelation of the user PII ($M(P^{\texttt{SP1\_REV}}_{\texttt{RECOVERED\_USER\_PII}}) \neq \emptyset$ in Figure 6, bottom right corner) in each revoking session must not be reached too. When `condActually` holds, we expect the number of decrypted UCHVE pieces to be greater or equal to $k$, and that the user's PII must eventually be revealed. Finally, we must ensure that the marking indicating illegal recovery of private VE key by the referees ($M(P^{\texttt{ALL\_REFEREES}}_{\texttt{REF\_RECOVERED\_VE\_PRIVATE\_KEY}}) \neq \emptyset$ in Figure 7, bottom left corner) is *not reachable*.

*Property 3 (Enforceable Conditions).* With these predicates and notations:

- $\mathsf{HasRefVEKey}\ M = (M(P^{\texttt{ALL\_REFEREES}}_{\texttt{REF\_RECOVERED\_VE\_PRIVATE\_KEY}}) \neq \emptyset)$
- $\mathsf{HasRecUsrPII}\ M = (M(P^{\texttt{SP1\_REV}}_{\texttt{RECOVERED\_USER\_PII}}) \neq \emptyset)$
- $\mathsf{HasRevocation}\ M = (M(P^{\texttt{SP1\_REV}}_{\texttt{SESSION\_BEING\_REVOKED}}) \neq \emptyset)$
- $k = 2, ..., n$ specifies the minimum number of referees who need to confirm the fulfilment of revocation conditions for a successful PII revocation
- $[M_0>$ is the set of reachable markings (from the initial marking $M_0$)

PIEMCP has *enforceable conditions* property if and only if $CPN_P^{M_0}$, with all the parameters in $A_{ses}$ being `false`, has the following behaviour:

- $\neg\textsc{Pos}(\mathsf{HasRefVEKey})$
- if $\neg$`condActually`, then
  - $\forall M \in [M_0>$: $\mathsf{HasRevocation}(M) \Rightarrow |M(P^{\texttt{SP1\_REV}}_{\texttt{UCHVE\_DECRYPT\_SUCCESS}})| < k$
  - $\forall\ r \in R$: $\neg\textsc{Pos}(\mathsf{Revoking}^r \wedge \mathsf{HasRecUsrPII}))$
- otherwise (`condActually`)
  - $\exists M \in [M_0>$: $\mathsf{HasRevocation}(M) \Rightarrow |M(P^{\texttt{SP1\_REV}}_{\texttt{UCHVE\_DECRYPT\_SUCCESS}})| \geq k$
  - $\forall\, r \in R$: $\textsc{Ev}(\mathsf{Revoking}^r \wedge \mathsf{HasRecUsrPII}))$ □

**Conditions Abuse Resistant.** The *conditions abuse resistant* property requires that an SP and an IdP must not be able to make the user to encrypt the PII or the VE private key, under a set of conditions different from those originally agreed. Similarly, an SP or IdP must not be able to successfully revoke the user's PII using conditions different from those originally agreed. Various attacks which may compromise this property have been modelled (`USER_ATTACK1`, `SP_ATTACK1`, `USER_ATTACK4`, `SP_ATTACK11`, `SP_ATTACK2`, and `SP_ATTACK3`). From the brief explanation of these attacks shown in Table 2, we can see that these attacks all involve manipulating the condition string at various stages of PIEMCP. The details of how we modelled these attacks are available in Appendix B (with the exception of `SP_ATTACK3` which have been explained in detail in Section 3.2).

We formalize this property as follows: When there are no attacks, the cryptographically bound conditions used to produce a VE ciphertext must be the same

as the one originally agreed (EqGenVEConds, EqSP1UchveConds, EqSP2UchveConds). When there are attacks targeting the general conditions used in the PE stage (parameterised by `USER_ATTACK1` and `SP_ATTACK1`), we expect that the IdP is able to detect such an attempt to use incorrect conditions (different from those originally agreed, which is captured by ¬EqGenVEConds) thus resulting in the incorrect encryption (UsrVE-F). Similar behaviour applies to the scenarios involving `USER_ATTACK4` and `SP_ATTACK11` as well as those involving `SP_ATTACK2`.

For attacks targeting the use of invalid conditions during the revocation stage (parameterised by `SP_ATTACK3`), we expect that the transition $T^{\texttt{SP1\_REV}}_{\texttt{USE\_ATTACK\_COND}}$ (Figure 6, middle-left) is not a dead transition anymore, and that the marking which indicates the revelation of user's PII (HasRecUsrPII), or the illegal revelation of VE private key (HasRefVEKey) should *not* be reached.

The following CPN pages (and the corresponding figures in which these pages are shown) are used in the property definition: `SETUP` (Appendix Figure 10), `USER_PE` (Figure 5), `USER_KE` (Appendix Figure 13), `USER_MC` (Appendix Figure 16), `IDP_PE` (Figure 4), `IDP_KE` (Appendix Figure 14), and `IDP_MC` (Appendix Figure 17).

*Property 4 (Conditions Abuse Resistant).* With these predicates and notations:

- HasGenCond $M = (M(P^{\texttt{SETUP}}_{\texttt{GEN\_COND}}) \neq \emptyset)$
- HasVECond $M = (M(P^{\texttt{USER\_PE}}_{\texttt{PII\_VE\_CIPHER}}) \neq \emptyset)$
- HasSP1Cond $M = (M(P^{\texttt{SETUP}}_{\texttt{SP1\_COND}}) \neq \emptyset)$
- HasSP1UchveCond $M = (M(P^{\texttt{USER\_KE}}_{\texttt{KVE\_UCHVE\_CIPHER}}) \neq \emptyset)$
- HasSP2Cond $M = (M(P^{\texttt{USER\_MC}}_{\texttt{SP2\_COND}}) \neq \emptyset)$
- HasSP2UchveCond $M = (M(P^{\texttt{USER\_MC}}_{\texttt{KVE\_UCHVE\_CIPHER}}) \neq \emptyset)$
- EqGenVEConds$(M, M') = (M(P^{\texttt{SETUP}}_{\texttt{GEN_COND}}) = M'(P^{\texttt{USER_PE}}_{\texttt{PII_VE_CIPHER}}))$
- EqSP1UchveConds$(M, M') = (M(P^{\texttt{SETUP}}_{\texttt{SP1_COND}}) = M'(P^{\texttt{USER_KE}}_{\texttt{KVE_UCHVE_CIPHER}}))$
- EqSP2UchveConds$(M, M') = (M(P^{\texttt{USER_MC}}_{\texttt{SP2_COND}}) = M'(P^{\texttt{USER_MC}}_{\texttt{KVE_UCHVE_CIPHER}}))$
- EqVECondIDP $M = (M(P^{\texttt{IDP\_PE}}_{\texttt{GEN\_COND}}) \neq \emptyset \ \wedge \ M(P^{\texttt{IDP\_PE}}_{\texttt{IDP\_VE\_CIPHER}}) \neq \emptyset \ \wedge$ $M(P^{\texttt{IDP\_PE}}_{\texttt{GEN\_COND}}) = M(P^{\texttt{IDP\_PE}}_{\texttt{IDP\_VE\_CIPHER}}))$
- EqUchve1CondIDP $M = (M(P^{\texttt{IDP\_KE}}_{\texttt{CIPHER\_UCHVE\_KVE}}) \neq \emptyset \ \wedge \ M(P^{\texttt{IDP\_KE}}_{\texttt{AGREED\_COND}}) \neq \emptyset \ \wedge$ $M(P^{\texttt{IDP\_KE}}_{\texttt{CIPHER\_UCHVE\_KVE}}) = M(P^{\texttt{IDP\_KE}}_{\texttt{AGREED\_COND}}))$
- EqUchve2CondIDP $M = (M(P^{\texttt{IDP\_MC}}_{\texttt{CIPHER\_UCHVE\_KVE}}) \neq \emptyset \ \wedge \ M(P^{\texttt{IDP\_MC}}_{\texttt{AGREED\_COND}}) \neq \emptyset \ \wedge$ $M(P^{\texttt{IDP\_MC}}_{\texttt{CIPHER\_UCHVE\_KVE}}) = M(P^{\texttt{IDP\_MC}}_{\texttt{AGREED\_COND}}))$
- UsrVE-F, UchveKE-T, and UchveKE-F, refer to definitions in Property 2
- HasRefVEKey and HasRecUsrPII, refer to definitions in Property 3
- $BE(T)$ is the set of all binding elements for a transition (instance) $T$
- $\forall M, M' \in [M_0>$, $\forall be \in BE$, $M \xrightarrow{be} M'$: $M'$ is reachable from $M$ upon firing $be$

PIEMCP has *conditions abuse resistant* property if and only if $CPN^{M_0}_P$ has the following behaviour:

- if all the parameters $A_{ses} \cup A_{rev}$ are `false`, then for each escrow session $s \in S$, and for markings $M, M' \in [M_0>$ such that Session$^s M$ and Session$^s M'$:
  - HasGenCond$M \wedge$ HasVECond$M' \Rightarrow$ EqGenVEConds$(M, M')$

  - $\mathsf{HasSP1Cond}M \wedge \mathsf{HasSP1UchveCond}M' \Rightarrow \mathsf{EqSP1UchveConds}(M, M')$
  - $\mathsf{HasSP2Cond}M \wedge \mathsf{HasSP2UchveCond}M' \Rightarrow \mathsf{EqSP2UchveConds}(M, M')$
- if `USER_ATTACK1` $\vee$ `SP_ATTACK1` (and others in $A_{ses} \cup A_{rev}$ are `false`), then
  $\exists\ s \in S$: $\text{Ev}(\mathsf{Session}^s \wedge \mathsf{UsrVE\text{-}F}) \wedge \neg\text{Pos}(\mathsf{Session}^s \wedge \mathsf{UsrVE\text{-}T} \wedge \mathsf{EqVECondIDP})$
- if `USER_ATTACK4` $\vee$ `SP_ATTACK11` (and others in $A_{ses} \cup A_{rev}$ are `false`), then
  $\exists\ s \in S$: $\text{Ev}(\mathsf{Session}^s \wedge \mathsf{UchveKE\text{-}F}) \wedge$
  $\neg\text{Pos}(\mathsf{Session}^s \wedge \mathsf{UchveKE\text{-}T} \wedge \mathsf{EqUchve1CondIDP})$
- if `USER_ATTACK2` (and others in $A_{ses} \cup A_{rev}$ are `false`), then
  $\exists\ s \in S$: $\text{Ev}(\mathsf{Session}^s \wedge \mathsf{UchveMC\text{-}F}) \wedge$
  $\neg\text{Pos}(\mathsf{Session}^s \wedge \mathsf{UchveMC\text{-}T} \wedge \mathsf{EqUchve2CondIDP})$
- if `SP_ATTACK3` (and others in $A_{ses} \cup A_{rev}$ are `false`), then
  - $\exists\ be \in BE(T^{\texttt{SP1\_REV}}_{\texttt{USE\_ATTACK\_COND}})$: $\exists_{M,M' \in [M_0>}[M \xrightarrow{be} M']$
    (i.e. $T^{\texttt{SP1\_REV}}_{\texttt{USE\_ATTACK\_COND}}$ is not a dead transition)
  - $\neg\text{Pos}(\mathsf{HasRefVEKey})$
  - $\exists\ r \in R$: $\neg\text{Pos}(\mathsf{Revoking}^r \wedge \mathsf{HasRecUsrPII})$ □

The above four property specifications have been implemented into ASK-CTL queries (based on the full syntax of ASK-CTL) in CPN Tools for model-checking the state spaces of $CPN_P^{M_0}$. The results of the execution of these queries over the 45 state spaces in total (capturing the protocol without attack or with various attacks, refer to Table 2) demonstrate that PIEMCP satisfies these four privacy compliance properties.

## 5 Related Work

Formal methods based on process algebra have been used to model and verify security protocols (such as LySa [8]). Process algebra allows the modelling of a system's behaviour as a set of algebraic statements. Common verification techniques used with process algebra include equational reasoning and model checking [4]. For example, the Pi-Calculus [16] supports *labeled transition* semantics in modelling a system. This allows the verification of protocols through state exploration techniques such as model checking. However, we choose not to use process algebra approach because of its complexity which tends to (unnecessarily) complicate even simple things [1]. In comparison to the graphical-based modelling approach in CPNs, the pi-calculus approach is a less intuitive approach to model a large distributed system such as PEPs. Model validation can only be performed by users who are experts in both the protocol itself *and* the process algebra syntax. Nevertheless, pi-calculus-based approach has been used to verify privacy-related technologies, such as the DAA protocol [3].

State exploration techniques (such as state space analysis and model checking) have also been widely used for security protocol analysis. Examples belonging to this category are Scyther [11], and ProVerif [7]. These are state-of-the-art tools capable of automatically detecting attacks in many security protocols. The main reason we do not use these tools is because the types of security properties

verifiable by these tools are not relevant to PEPs. Instead, they are mostly relevant to authentication and key agreement protocols, i.e. *secrecy*, *authenticity*, and their variants. When protocols related to privacy are verified using these tools, the privacy property is reduced to confidentiality and authenticity. We argue that this is a simplistic approach to verifying privacy and that privacy does not simply equate to confidentiality and/or authenticity. The *behaviour* of a protocol in preserving or violating a user's privacy is just as important. These tools also lack the rich graphical and simulation support of CPNs.[7] Therefore, we do not find these tools to be suitable for our purpose. Although CPNs have been widely used to analyze industrial communication protocols (such as Transmission Control Protocol (TCP) [6]), its use in the area of security protocols is still new with limited documented cases. For example, Al-Azzoni et al [2] used CPN to model and verify the Tatebayashi, Matsuzaki, and Newman (TMN) key exchange protocol [23]. The main difference between our work and theirs is that they focus on verifying the *secrecy* property of the TMN protocol, while our work focus on verifying the privacy behaviour of PEPs. The work presented in this paper is an extension of our earlier work [22]. The main differences include: (1) the improvement of the PIEMCP CPN model by re-structuring the model in terms of modularisation of individual entities, their communication channels, and different stages of operations; (2) the inclusion of the dynamic referee behaviour, i.e. the `ALL_REFEREES` page and instantiation of the one `REFEREE` page according to the number of referees involved; (3) a detailed analysis of the attack scenarios, which leads to the finding of a set of necessary configurations of the PIEMCP CPN model capturing all possible attack behaviours; (4) the elaboration of privacy compliance properties in terms of an improved formalisation of property definitions which we believe is more precise and fine-grained (e.g. each property is now defined in terms of a set of relevent attack behaviours, instead of a "blanket" approach used in the previous work [22]); and (5) analysis and verification of PIEMCP based on the updated CPN model and privacy compliance property definitions.

## 6 Conclusion

We have shown that CPNs can be used to model complex PEPs, a class of cryptographic protocols, and support the verification of their privacy compliance properties based on state space analysis. We have also proposed several modelling techniques, notably the cryptographic primitive abstraction (capturing complex primitives and zero-knowledge proof protocol) and parameterised attacks. We have also shown how we can formalise and verify privacy compliance properties using standard state space analysis techniques and ASK-CTL queries.

Future work involves refinement and generalization of the modelling and analysis approaches proposed in this paper such that they can be applied to other PEPs. We also hope to build a better user front-end to simplify and automate the

[7] Scyther provides some static graphical support. However, it falls short of interactive protocol simulation and graphically-driven protocol specification.

tasks required in the modelling and verification of PEPs. The function of such a front-end could be as simple as aiding users with the configuration of attack parameters without the need of knowing CPNs. Another long-term goal is to achieve automated attack detections for PEPs using a CPN-based approach.

## Acknowledgements

This work is partially supported by National ICT Australia (NICTA). NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

## References

1. van der Aalst, W.: Pi calculus versus Petri nets: Let us eat humble pie rather than further inflate the Pi hype. BPTrends pp. 1–11 (May 2005)
2. Al-Azzoni, I., Down, D.G., Khedri, R.: Modeling and verification of cryptographic protocols using Coloured Petri nets and Design/CPN. Nordic Journal of Computing 12(3), 201–228 (2005)
3. Backes, M., Maffei, M., Unruh, D.: Zero-knowledge in the applied Pi-calculus and automated verification of the direct anonymous attestation protocol. In: IEEE Symposium on Security and Privacy. pp. 202 –215 (May 2008)
4. Baeten, J.C.M.: A brief history of process algebra. Theor. Comput. Sci. 335(2-3), 131–146 (2005)
5. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: ACM CCS. pp. 62–73 (1993)
6. Billington, J., Han, B.: Modelling and analysing the functional behaviour of TCP's connection management procedures. STTT 9(3-4), 269–304 (2007)
7. Blanchet, B.: An efficient cryptographic protocol verifier based on Prolog rules. In: 14th IEEE CSFW. pp. 82–96. IEEE Computer Society (2001)
8. Bodei, C., Buchholtz, M., Degano, P., Nielson, F., Nielson, H.R.: Static validation of security protocols. J. Comput. Secur. 13(3), 347–390 (2005)
9. Brickell, E.F., Camenisch, J., Chen, L.: Direct anonymous attestation. In: Atluri, V., Pfitzmann, B., McDaniel, P.D. (eds.) ACM CCS. pp. 132–145. ACM (2004)
10. Christensen, S., Mortensen, K.H.: Design/CPN ASK-CTL Manual - Version 0.9. University of Aarhus, Aarhus C, Denmark (1996)
11. Cremers, C.J.: The scyther tool: Verification, falsification, and analysis of security protocols. In: CAV '08. pp. 414–418. Springer-Verlag, Berlin, Heidelberg (2008)
12. Dolev, D., Yao, A.C.C.: On the security of public key protocols. IEEE Transactions on Information Theory 29(2), 198–207 (1983)
13. Gilmore, S.: Programming in standard ML '97: A tutorial introduction. Tech. rep., The University of Edinburgh (1997)
14. Jensen, K., Kristensen, L.M.: Coloured Petri Nets - Modelling and Validation of Concurrent Systems. Springer (2009)
15. Koblitz, N., Menezes, A.: Another look at "provable security". J. Cryptology 20(1), 3–37 (2007)
16. Milner, R.: Communicating and Mobile Systems: the Pi-Calculus. Cambridge University Press (June 1999)

17. Ngo, L., Boyd, C., Nieto, J.: Automating computational proofs for public-key-based key exchange. In: Provable Security, LNCS, vol. 6402, pp. 53–69. Springer (2010)
18. Pointcheval, D.: Contemporary cryptology - Provable security for public key schemes. pp. 133–189. Advanced Courses in Mathematics, Birkhäuser (2005)
19. Suriadi, S.: Strengthening and Formally Verifying Privacy in Identity Management Systems. Ph.D. thesis, Queensland University of Technology (September 2010)
20. Suriadi, S., Foo, E., Josang, A.: A user-centric federated single sign-on system. Journal of Network and Computer Applications 32(2), 388–401 (March 2009)
21. Suriadi, S., Foo, E., Smith, J.: Private information escrow bound to multiple conditions. Tech. rep., Information Security Institute - Queensland University of Technology (2008), `http://eprints.qut.edu.au/17763/1/c17763.pdf`
22. Suriadi, S., Ouyang, C., Smith, J., Foo, E.: Modeling and verification of privacy enhancing protocols. In: Formal Methods and Software Engineering, LNCS, vol. 5885, pp. 127–146. Springer Berlin / Heidelberg (2009)
23. Tatebayashi, M., Matsuzaki, N., et al.: Key distribution protocol for digital mobile communication systems. In: CRYPTO '89. pp. 324–334. Springer (1989)
24. WP 14.1: PRIME (Privacy and Identity Management for Europe) - Framework V3 (March 2008)

# A Description of Relevant PIEMCP CPN Model

In this section, relevant PIEMCP CPN pages which are used in this paper are shown and described.

## A.1 Second-level Pages

To provide the overall picture of the PIEMCP CPN model, the second-level pages of SP1 and IDP (which have not been shown in the main paper) are displayed in Figure 8 and Figure 9 respectively. Note that since SP2 is only involved in the MC stage, the second-level page of SP2 is at the same level of granularity as the third-level pages of other entities. The details of the SP2's second level page are provided in Section A.5.

## A.2 Setup Page

Figure 10 depicts the setup phase that runs at the user side. It is used to generate the necessary condition string as well as to control the number of escrow session to be executed by the model. The transition `SETUP'USER_GENERATES_CONDITIONS` is the very first transition that is executed in an escrow session, and its transition guard ensures that the transition can only be enabled if the number of session counter `counter` is less or equal to the model parameter `session` (see Table 1).

If the transition is enabled, the user then starts generating the necessary *agreed* condition strings between user and SP1. Note that in Section 2 and in Figure 1, it is depicted that only one condition string is used between user and SP1; however, the protocol actually requires another condition string that needs to be used to generate VE ciphertext (called *GenCond*). This information has

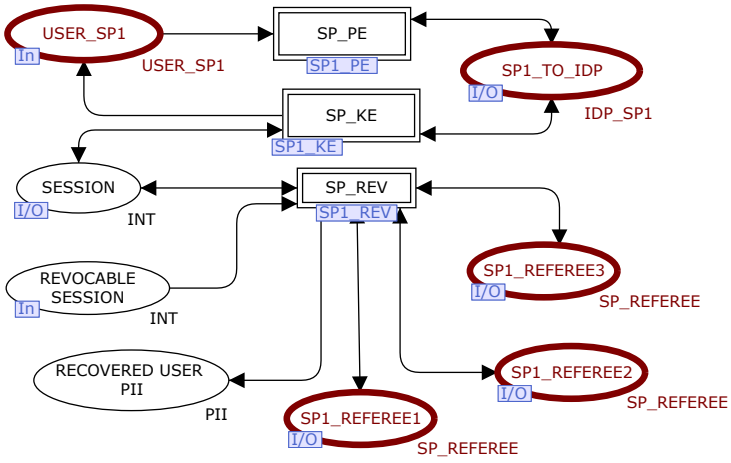


Fig. 8: SP1 Second Level Page

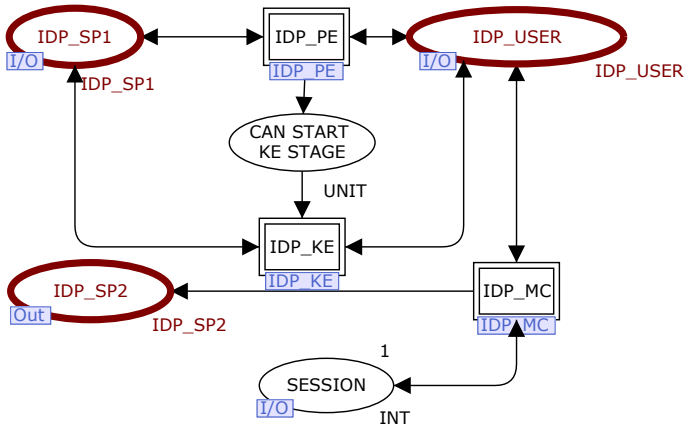


Fig. 9: IDP Second Level Page

been omitted in the main paper to simplify the protocol description. Furthermore, the agreement process between SP1 and user in the condition string has been abstracted out from the model as our model assumes that both parties know the agreed conditions before the start of the PE stage.

### A.3 PE Pages

The USER PE page and IDP PE page have been described in the main paper (Section 3.2), therefore, in this section, we only describe the SP1 PE page as shown on Figure 11, as well as a fourth-level page which is used inside the USER PE page (see Figure 12).

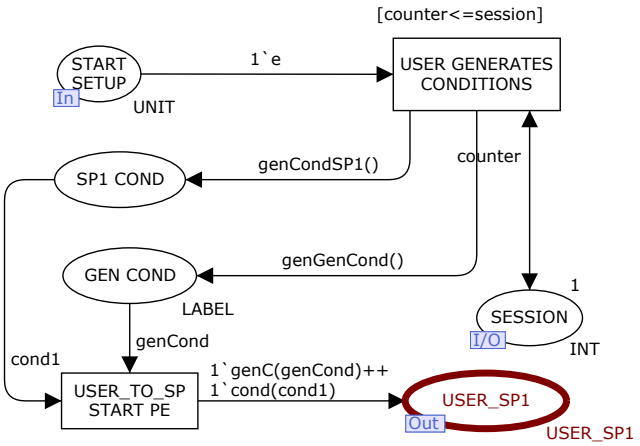


Fig. 10: User SETUP Page

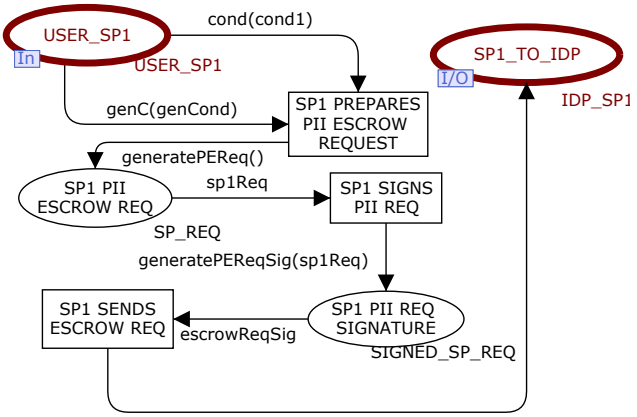


Fig. 11: SP1 PE Page

Figure 11 shows the operation of SP1 as it receives a service request from the user. It prepares the PII escrow request (see transition `SP1_PE'SP1_PREPARES_PII_ESCROW_REQUEST`), signs the request (see transition `SP1_PE'SP1_SIGNS_PII_REQ`) and sends the request to the IdP. This request then triggers various operations involved during both PE and KE stage. SP1 will receive a response to this request at the end of the KE stage (the details of which are described in the next section).

Figure 12 shows the operation of the user who generates a per-session data, including a one-time `pubk_ve` and `privk_ve` key pair, as well as the corresponding VE ciphertext of the PII.

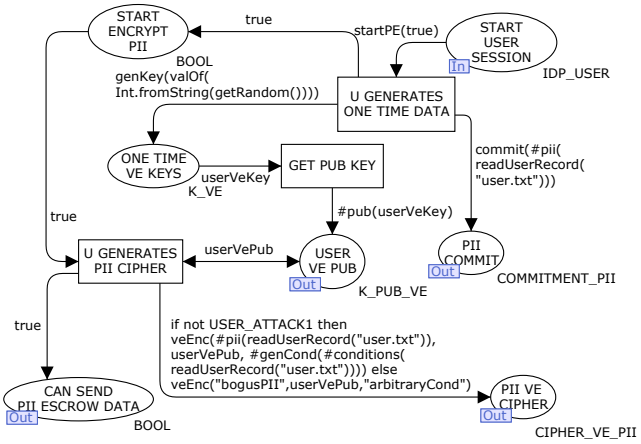


Fig. 12: A fourth-level page GENERATE_ONE_TIME_DATA Page

### A.4 KE Pages

Figure 13 and Figure 14 show the PIEMCP CPN Model of the USER and IDP KE stage. Upon receiving the request to start the DAA from the IDP, the user and IDP engage in `PK-DAA` protocol (`NT-KE-1` on Figure 1). This operation requires the user's TPM to generate some proof which consists of (1) a set of attestation identity key (AIK) keys which are then signed using the corresponding DAA key (see `USER_KE'BLINDED_AIK_PUB_KEY_SIG`, and a set of session keys which are signed using the corresponding AIK key (see `USER_KE'SESSION_KEY`), and a blinded representation of the original DAA signature (see `USER_KE'DAA_SIGNATURE`).[8]. This operation is captured as code-region within the transition `USER_KE'TPM_GENERATES_AIK_KEY_AND_SESSION_KEY`. This proof is then sent to the IdP.

Upon receiving the DAA proof, the IDP then verifies the proof (see Figure 14) (captured as a code-region attached to the transition `IDP_KE'VERIFY_PKDAA_PROOF`. Assuming the verification operation returns a "true" value, the IDP then signals the user to generate UCHVE pieces (see Section 2 for details).

The generation of the UCHVE pieces and the corresponding TPM Proof (`NT-KE-2`) are captured by the transitions `USER_KE'TPM_EXECUTES_MODULE_2` and `USER_KE'TPM_GENERATES _CORRECT_EXECUTION_PROOF` in Figure 13. These messages are then sent to the IdP who will verify their correctness as captured by the transition `IDP_KE'IDP_VERIFIES_MODULE_2_TPM_PROOF` in Figure 14. Upon a successful verification of the generated UCHVE pieces, the IdP then generates a response to be sent to SP1 (`NT-KE-3`).

Upon receiving the message from IdP, SP1 then contacts user to indicate the completion of the KE stage (see Figure 15). SP1 also stores the session data (captured as a code-region for the transition `SP1_KE'SEND_RESPONSE` in Figure **??**). Once the user receives such a confirmation, theoretically, the user can start

[8] Readers who are interested in the details of the DAA protocol should refer to [9]

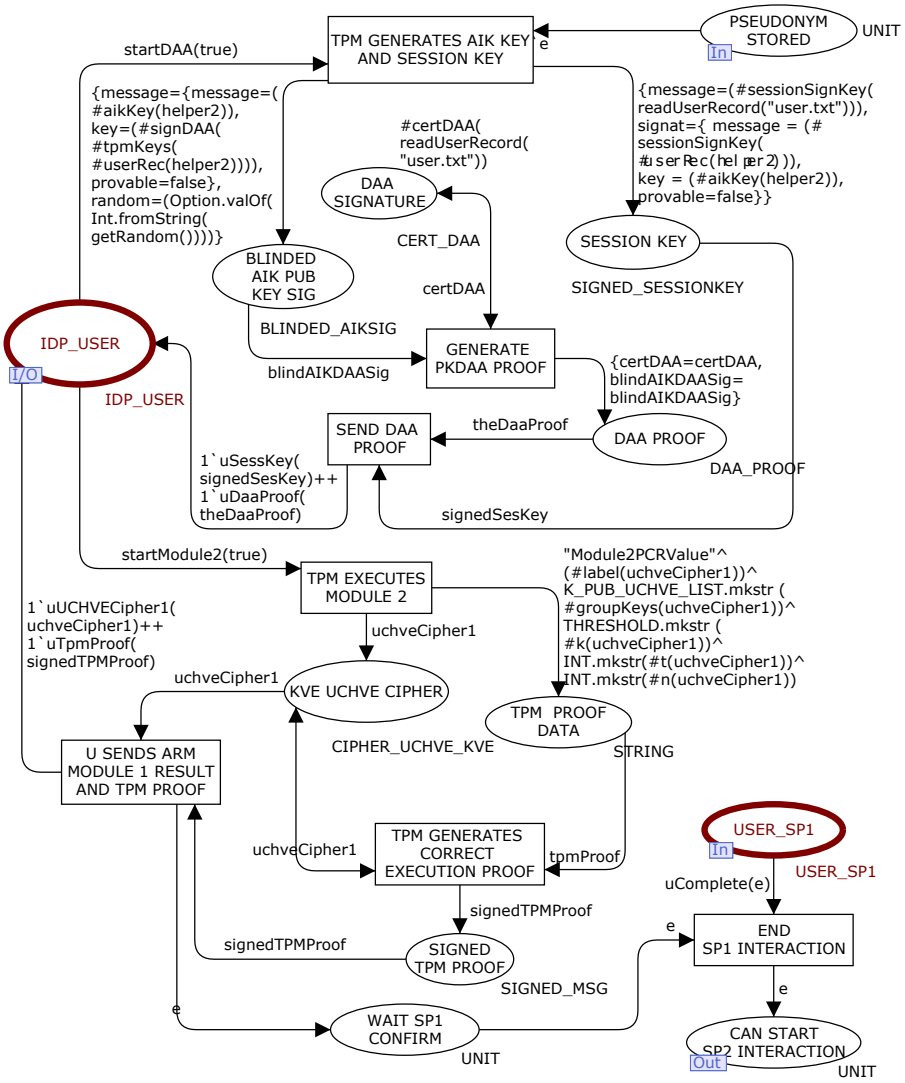


Fig. 13: PIEMCP CPN USER KE Page

receiving services from SP1; however, we are not interested in the actual services being consumed by the user, therefore, in our model (see the bottom part of Figure 13), we abstract out such an interaction and simply models the situation whereby the user ends the interaction with SP1 and ready to request service from another service provider (SP2).

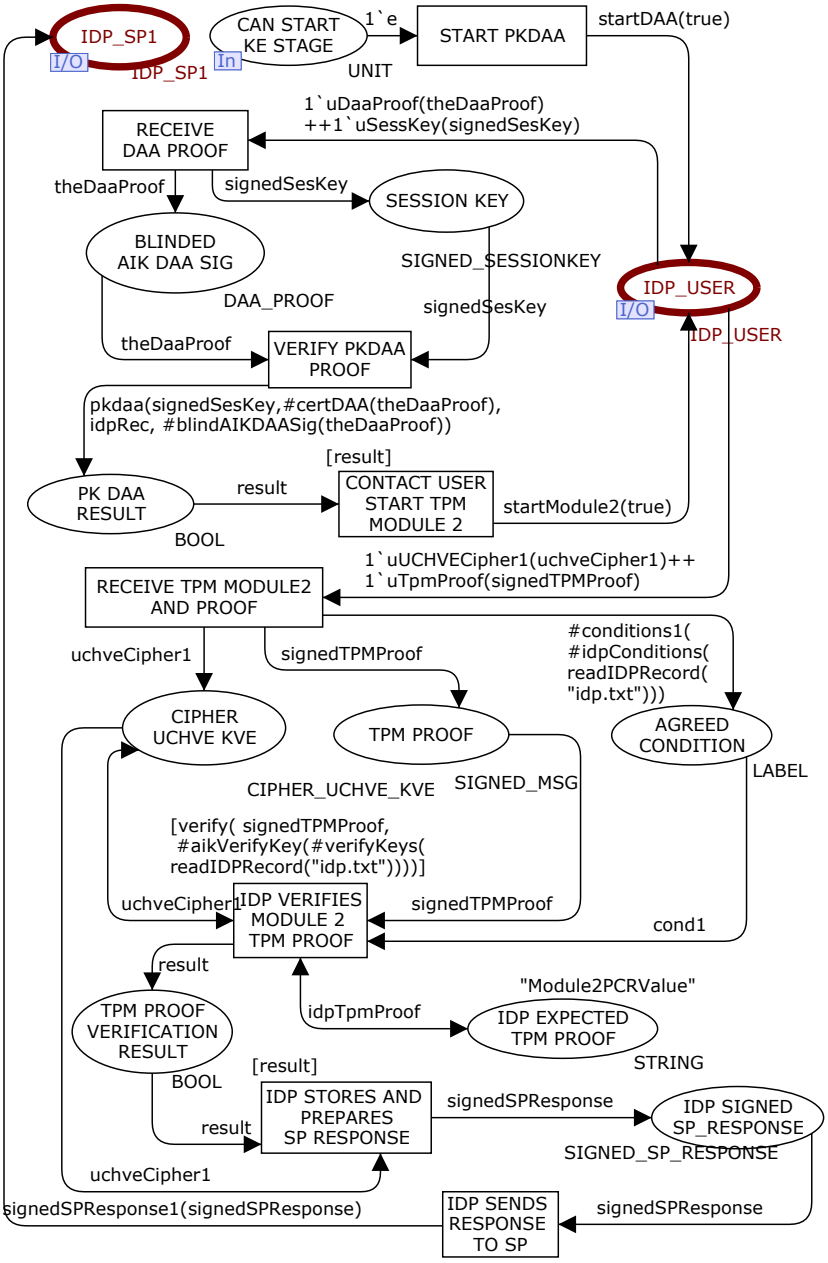


Fig. 14: PIEMCP CPN IDP KE Page

### A.5 MC Pages

Figure 16, Figure 17, and Figure 18 show the MC-stage pages for user, IDP, and SP2 respectively. Upon the completion of the KE Stage, the user then proceeds

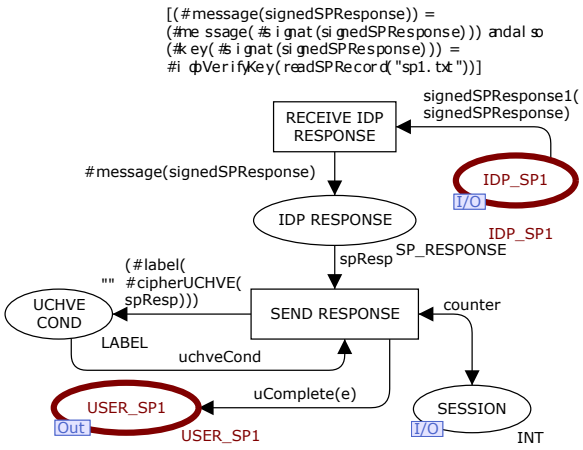


Fig. 15: PIEMCP CPN SP1 KE Page

to SP2 to request some services (see transition `USER_MC'SENC_REQUEST_TO_SP2` in Figure 16) in which another set of condition string is included.

SP2, upon receiving this message, then proceeds to request for user's PII to be escrowed under another set of conditions (agreed between the user and SP2) to the IdP via redirection through the user (see transition `SP2'SP2_REDIRECTS_ESCROW_REQ_TO_IDP_VIA_USERS` in 17). Again, the process of agreeing on a set of conditions has been abstracted out from the model and it is assumed that both parties always know the agreed set of conditions prior to the start of the MC session. User then forwards the request to the IdP, and at the same time, provide the session pseudonym that the IdP gave earlier at the completion of PE stage (see transition `USER_MC'USER_ FORWARDS_SP2_REQ_WITH_SESSION_PSEUDONYM)` which corresponds to (`NT-MC-1` message in Figure 1 - note that such a redirection technique has been omittted in Figure 1 for simplicity).

IDP then checks if such a pseudonym exists and that its corresponding session is still active. If so, it then contacts the user (see transition to provide another set of UCHVE pieces, this time encrypted under the condition string agreed with SP2 (`NT-MC-2` message). This effectively triggers the execution of another TPM provable execution Module 2 (see `IDP_MC'IDP_STARTS_MODULE_2_WITH_USER` in Figure 17).

The user then executes the TPM Module 2 in order to generate the UCHVE pieces, and the corresponding TPM correct execution is also generated. Message `NT-MC-3` (from Figure 1 is then sent to the IdP (see transition `USER_MC'U_SENDS_TO_IDP_MODULE_2_RESULT_AND_TPM_PROOF` in Figure 16).

Once the IdP verifies the correctness of the UCHVE pieces, it indicates the end of the escrow session on the IdP side: it prepares a response to SP2 and stores the session data (see transition `IDP_MC'IDP_STORES_AND _PREPARES_SP_RESPONSE` in Figure 17). IdP then sends the response to the SP2 (message `NT-MC-4` in Figure 1).

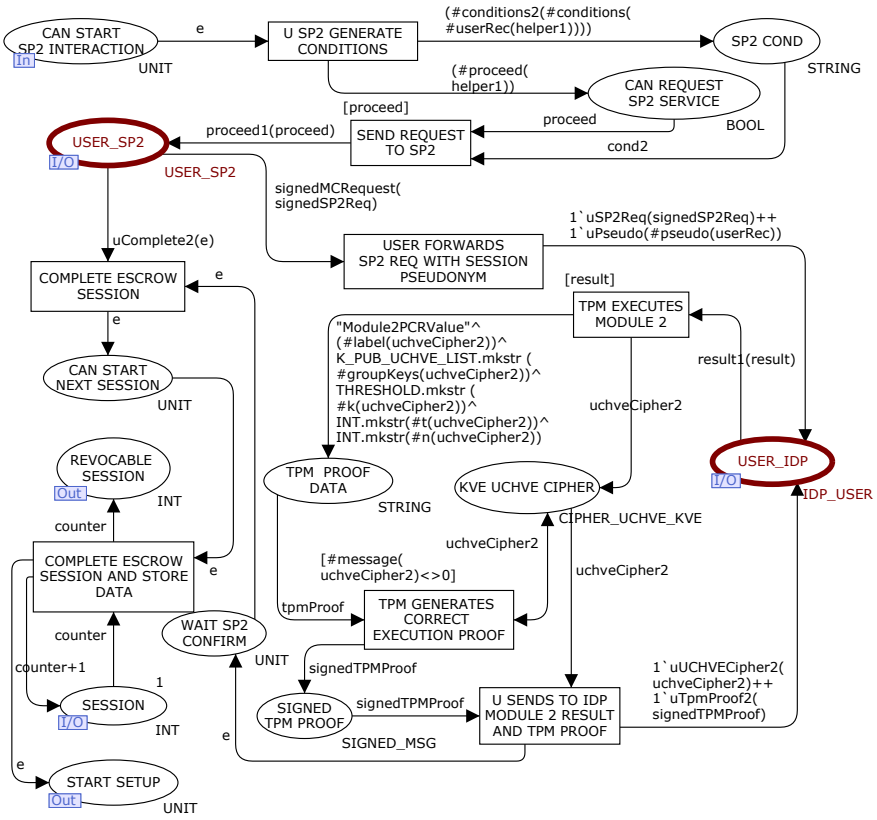


Fig. 16: PIEMCP USER MC Page

SP2 then verifies the IdP response and if all verification checks pass, stores the information and sends a signal to user to confirm that the user's PII has been successfully escrowed. The receipt of the confirmation from SP2 marks the end of the escrow session for the user, thus, it stores the session data (see transition `USER_MC'COMPLETE_ESCROW_SESSION_AND_STORE_DATA` in Figure 16). The purpose of the storage of session data is to facilitate session data analysis such that *state-independent* properties (that is, those properties which are not dependent on the state and other behavioural information of the model) may be analyzed or verified following the simulation of the model. For example, through session data analysis, one may be able to analyze if there exists any protocol data that may link a user to multiple escrow sessions, hence, may jeopardise his/her privacy. Such an analysis is not detailed in this paper, but an example of which can be obtained from [19]. The session data are also used during the revocation stage to retrieve necessary escrow data needed to revoke a user's PII.

Furthermore, the end of one escrow session also means that the user can now start the next escrow session if he/she wishes to. Therefore, in our model, we

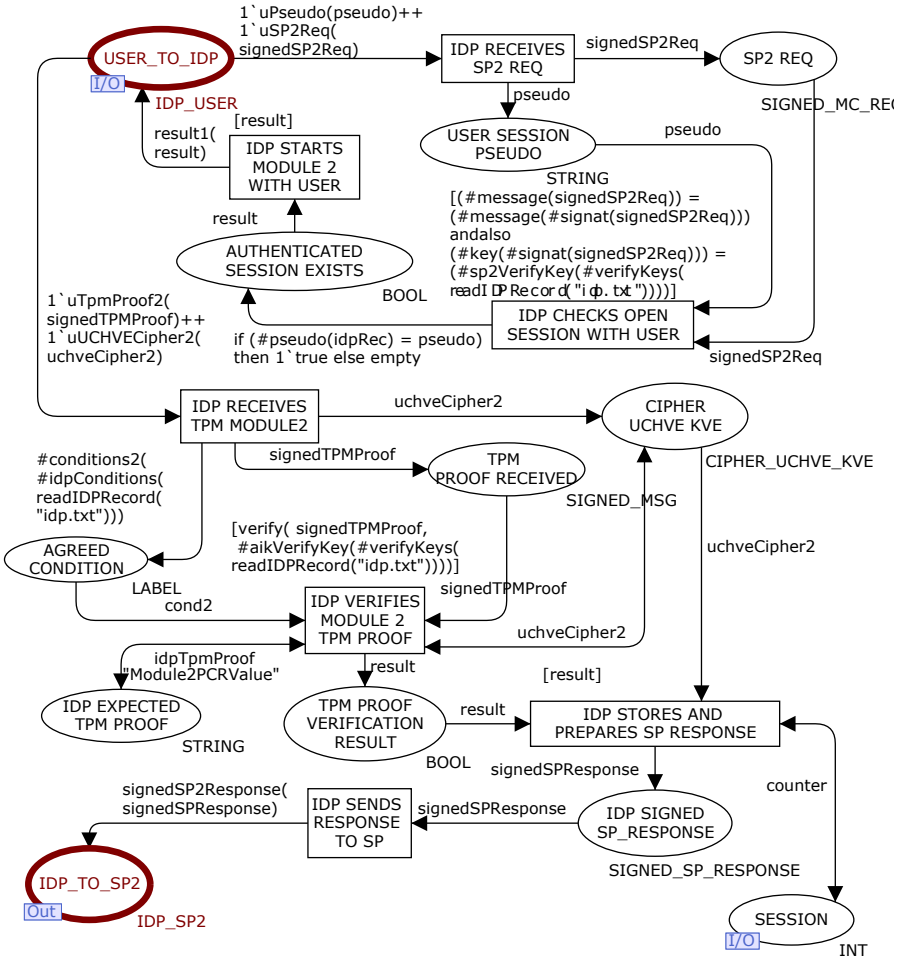


Fig. 17: PIEMCP IDP MC Page

allow such a repeat of the escrow session by increasing the session counter by one and thus preparing the model to execute another escrow session (that is, the place `USER_MC'CAN_START_NEXT_SESSION` will be populated with a token). The decision on whether another round of escrow session should be executed is determined in the `SETUP` page (see Section A.2).

## B Modelling Attacks

In this section, our approach in configuring the model to capture various types of attacks is detailed. Those attacks whose modelling have been described in the main part of the paper are not repeated here. These attacks include `USER_ATTACK2`

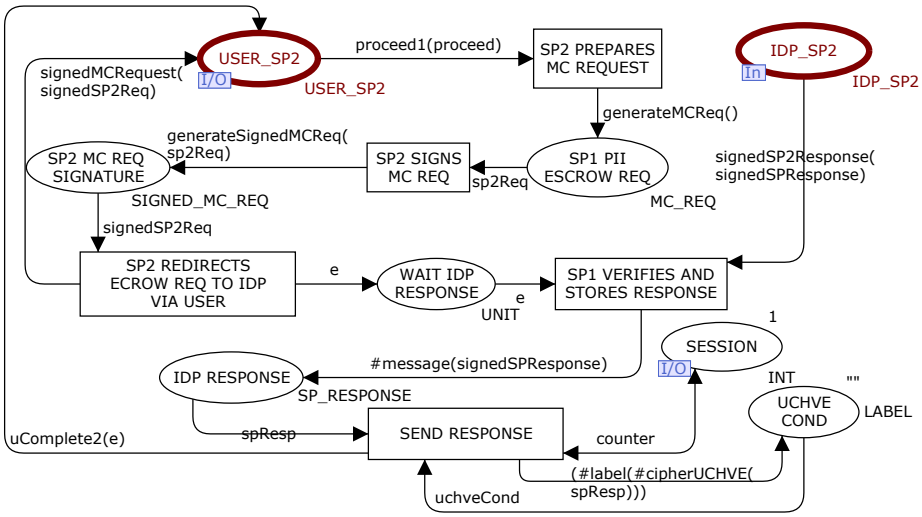


Fig. 18: PIEMCP SP2 Page

(explained on pp. 12), `SP_ATTACK3`, `REF_ATTACK1`, and `REF_ATTACK2` (the last three attacks are detailed from pp. 12 onwards).

### B.1 Modelling Attacks by User

The `USER_ATTACK1` parameter captures the behaviour of a user who attempts to, during the PE stage, generate a VE ciphertext containing incorrect PII and under condition *ArbitraryCond* which is not the same with the agreed conditions wiht SP1 (that is, $Cond1 \neq ArbitraryCond$). This malicious behaviour is captured on the fourth-level page `GENERATE_ONE_TIME_DATA` shown on Figure 12 in the arc inscription from the transition `GENERATE_ONE_TIME_DATA'U_GENERATES_PII_CIPHER` to the place `GENERATE_ONE_TIME_DATA'PII_VE_CIPHER`. This arc inscription states that if the `USER_ATTACK1` parameter is switched off, then the user will generate a VE ciphertext containing correct PII and conditions (as read from the user's record); otherwise, random PII values and conditions are used. The notion of "arbitrary condition" and random PII values is captured by simply using a condition string which is not the same as the value originally generated by the user in Figure 10 during the SETUP phase and by using the PII values that are not the same as the one stored in the user's session data (which is a value of `m_a`).

The `USER_ATTACK3` parameter captures the behaviour of a user who attempts to provide an incorrect UCHVE parameters during the execution of `TPM Module 2`. The `USER_ATTACK4` parameter captures the behaviour of a user who attempts to use an incorrect condition in the generation of the UCHVE ciphertext during the execution of `TPM Module 2`. To capture these attack behaviours, we use a code-region linked to the transition `USER_KE'TPM_EXECUTES_MODULE_2` (see Figure 13 for the transition). The code-region is detailed in Table 3. The notion

of incorrect UCHVE parameters and condition are captured by using a set of UCHVE parameters (inluding the value of $k$, $t$, $n$, and $\texttt{privk}_{VE}$) which are *different* from what have been agreed upon (as reflected in the user's session data record: $k$=2, $|t|$=2, $|n|$=3, ).

```
Code-region for transition USER_KE'TPM_EXECUTES_MODULE_2
=======================================================
input ();
output (uchveCipher1);
action
let
 val userRec = readUserRecord("user.txt")
 val groupKeys = if not USER_ATTACK3 then #uchvePubKeys(userRec)
                 else [7,8,9]
 val msg = if not USER_ATTACK3 then #priv(#veKeys(userRec)) else 0
 val desigMembers = if not USER_ATTACK3 then #desigMembers(userRec) else [7,8,9]
 val k = if not USER_ATTACK3 then #k(userRec) else 99
 val t = if not USER_ATTACK3 then length (#desigMembers(userRec)) else 99
 val n = if not USER_ATTACK3 then length (#uchvePubKeys(userRec)) else 99
 val label = if not USER_ATTACK4 then #conditions1(#conditions(userRec)) else
             "hardToFulfillConditions"

 val pub = #pub(#veKeys(userRec))
 val correct = msg = pub

 val uchveCipher1 = uchveEnc(msg, groupKeys, correct, desigMembers, t, k, n, label)
in
 uchveCipher1
end;

Function definition of 'uchveEnc'
=================================
fun uchveEnc(msg:K_PRIV_VE, groupKeys:K_PUB_UCHVE_LIST, correct:BOOL,
             desigMembers:DESIG_MEMBERS_LIST, t:INT, k:THRESHOLD, n:INT,
             label:LABEL) =
            {message=msg, groupKeys=groupKeys, desigMembers=desigMembers,
             t=t,k=k, n=n,label=label,provable=true};
```

Table 3: Code-region and function capturing USER_ATTACK3 and USER_ATTACK4

### B.2 Modelling Attacks by SP1 and SP2

The SP_ATTACK1 and SP_ATTACK11 parameters captures the behaviour of a malicious SP1 who attempts to send an incorrect conditions to the IdP during PE and KE stage. As explained in Section A.2, there are actually two types of condition strings used during user's interaction with SP1: the *GenCond* used during the PE stage and *Cond1* used during the KE stage (although we abstracted out the *GenCond* parameter in Figure 1 for simplicity). The SP_ATTACK12 captures the behaviour of a malicious SP1 who attempts to provide IdP with a set of UCHVE parameters which are different from what were agreed with the user.

All these three attacks are captured in the SP1_PE page, as the outgoing arc inscription from the transition SP1_PE'SP1_PREPARES_PII_ESCROW_REQUEST - see Figure 11. Because of the length of the arc inscription required to capture

these attacks, we have instead captured the required inscription as a function to improve readability of the model. This function, called `generatePEReq()`, is detailed in Table 4. The notion of incorrect condition and UCHVE parameters is the same as the one used in Section B.1.

```
Details of function 'generatePEReq()'
=====================================
fun generatePEReq()=
if SP_ATTACK1 then
{genCond="EasyGenCond",
conditions1=(#conditions(readSPRecord("sp1.txt"))),
uchvePubKeys=(#groupKeys(#cipherUCHVE(readSPRecord("sp1.txt")))),
k=(#k(#cipherUCHVE(readSPRecord("sp1.txt")))),
t=(#t(#cipherUCHVE(readSPRecord("sp1.txt")))),
n=(#n(#cipherUCHVE(readSPRecord("sp1.txt"))))} else

if SP_ATTACK11 then
{genCond=(#genCond(readSPRecord("sp1.txt"))),
conditions1="EasyToFulfillConditions",
uchvePubKeys=(#groupKeys(#cipherUCHVE(readSPRecord("sp1.txt")))),
k=(#k(#cipherUCHVE(readSPRecord("sp1.txt")))),
t=(#t(#cipherUCHVE(readSPRecord("sp1.txt")))),
n=(#n(#cipherUCHVE(readSPRecord("sp1.txt"))))} else

if SP_ATTACK12 then
{genCond=(#genCond(readSPRecord("sp1.txt"))),
 conditions1=(#conditions(readSPRecord("sp1.txt"))),
uchvePubKeys=[7,8,9],
k=1, t=1, n=3} else

{genCond=(#genCond(readSPRecord("sp1.txt"))),
 conditions1=(#conditions(readSPRecord("sp1.txt"))),
uchvePubKeys=(#groupKeys(#cipherUCHVE(readSPRecord("sp1.txt")))),
k=(#k(#cipherUCHVE(readSPRecord("sp1.txt")))),
t=(#t(#cipherUCHVE(readSPRecord("sp1.txt")))),
n=(#n(#cipherUCHVE(readSPRecord("sp1.txt"))))};
```

Table 4: Details of the function `generatePEReq()` capturing `SP_ATTACK1`, `SP_ATTACK11`, and `SP_ATTACK12` attacks

The `SP_ATTACK2` and `SP_ATTACK22` parameters capture the same attack behaviour as `SP_ATTACK11` and `SP_ATTACK12` respectively, except that the former two attacks happen in the interaction between the user and SP2 during the MC stage. These two attacks are therefore modelled in the **SP2!** (**SP2!**) page in the outgoing arc inscription of the transition `SP2'SP2_PREPARES_MC_REQUEST`. This arc inscription contains a function `generateMCReq()` whose details are provided in Table 5.

The `SP_ATTACK6` parameter captures the behaviour of a malicious SP2 who attempts to use an invalid signature key to sign the `NT-MC-1` message (see Figure 1). This attack is captured in the outgoing arc inscription (as a function called `generateSignedMCReq(sp2Req)`) of the transition `SP2'SP2_SIGNS_MC_REQ` - see Figure 18. The details of this function are provided in Table 5.

Finally, the `SP_ATTACK7` represents the behaviour of malicious SP1 and SP2 who collide to use the same condition string between the KE and MC stage.

```
Details of the function 'generateMCReq()'
=========================================
fun generateMCReq()=
if SP_ATTACK2 then
{conditions2="EasyToFulfillConditions",
uchvePubKeys=(#groupKeys(#cipherUCHVE(readSPRecord("sp2.txt")))),
k=(#k(#cipherUCHVE(readSPRecord("sp2.txt")))),
t=(#t(#cipherUCHVE(readSPRecord("sp2.txt")))),
n=(#n(#cipherUCHVE(readSPRecord("sp2.txt"))))} else

if SP_ATTACK22 then
{conditions2=(#conditions(readSPRecord("sp2.txt"))),
uchvePubKeys=[7,8,9],
k=1, t=1, n=3} else

{conditions2=(#conditions(readSPRecord("sp2.txt"))),
uchvePubKeys=(#groupKeys(#cipherUCHVE(readSPRecord("sp2.txt")))),
k=(#k(#cipherUCHVE(readSPRecord("sp2.txt")))),
t=(#t(#cipherUCHVE(readSPRecord("sp2.txt")))),
n=(#n(#cipherUCHVE(readSPRecord("sp2.txt"))))};

Details of the function 'generateSignedMCReq(sp2Req:MCReq)'
===========================================================
fun generateSignedMCReq(sp2Req:MC_REQ)=
if not SP_ATTACK6 then 1`{message=sp2Req,
signat={message=sp2Req,
key=(#signKey(readSPRecord("sp2.txt"))),
provable=false}} else
1`{message=sp2Req,
signat={message=sp2Req,
key=999, provable=false}};
```

Table 5: Details of the function `generateMCReq()` capturing `SP_ATTACK2` and `SP_ATTACK22`, and the function `generateSignedMCReq(sp2Req:MC_REQ)` capturing `SP_ATTACK6`.

This is an attack because condition used between KE, MC, and subsequent MC stages within an escrow session should be specific to a particular SP to whom the user is interacting. Sharing the same condition string means that the fulfillment of one will allow all other SPs within the same session to receive the user's PII information (which may be a violation of user's privacy).

This attack is captured on the code-region linked to the transition `USER_MC'U_SP2_GENERATE_CONDITIONS` (see Figure 16). The details of this code-region are provided in Table 6. If this attack parameter is switched on, the condition string to be used will then be the same condition string as the user used with SP1. This code-region implicitly captures the sharing of data between SP1 and SP2 (otherwise, SP2 will not be able to learn the condition string that SP1 used earlier).

```
input();
output (helper1);
action
let
val condRandom = getRandom()
val userRec = readUserRecord("user.txt")
val userRec1 = readUserRecord("user_sess1.txt")
val userRec2 = readUserRecord("user_sess2.txt")
 val sp1Rec = readSPRecord("sp1.txt")
val condSP1 = #conditions(sp1Rec)
val condSP2 = if SP_ATTACK7 then condSP1 else "Conditions SP2"^condRandom
val conditions = #conditions(userRec)
val conditions1 = #conditions(userRec1)
val conditions2 = #conditions(userRec2)
val proceed = validCond(condSP2, conditions1, conditions2, conditions)
 val conditions = USER_CONDITIONS.set_conditions2 conditions condSP2
 val userRec = USER_RECORD.set_conditions userRec conditions
 val userRec = updateUserRecord("user.txt", userRec)
 val spRec = readSPRecord("sp2.txt")
 val spRec = SP_RECORD.set_conditions spRec condSP2
 val spRec = updateSPRecord("sp2.txt", spRec)
in
  createHelper1(proceed, readUserRecord("user.txt"))
end;
```

Table 6: Details of the code-region for the transition `USER_MC'U_SP2_GENERATE_CONDITIONS` to capture `SP_ATTACK7`.